Propositions pour améliorer l'élasticité et la robustesse d'une fonction de routage NDN virtualisée

Cédric Enclos

Mémoire d'ingénieur

Propositions pour améliorer l'élasticité et la robustesse d'une fonction de routage NDN virtualisée

Cédric Enclos

Année 2014–2015

Stage de fin d'études réalisé dans l'entreprise LORIA

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Maître de stage : Thibault Cholez

Encadrant universitaire : Jean-Marie Moureaux

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Enclos, Cédric

Élève-ingénieur(e) régulièrement inscrit(e) en 3e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31213324

Année universitaire : 2014–2015

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Propositions pour améliorer l'élasticité et la robustesse d'une fonction de routage NDN virtualisée

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Villers-lès-Nancy, le 23 septembre 2015

Signature :

Mémoire d'ingénieur

Propositions pour améliorer l'élasticité et la robustesse d'une fonction de routage NDN virtualisée

Cédric Enclos

Année 2014–2015

Stage de fin d'études réalisé dans l'entreprise LORIA

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Cédric Enclos
560, route d'Epinal
88400 Gérardmer
06 28 36 49 21
cedric.enclos@telecomnancy.eu

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

LORIA
615, rue du Jardin Botanique
54500 Villers-lès-Nancy
+33 (0)3 83 59 20 00

Laboratoire lorrain de recherche en informatique et ses applications

Maître de stage : Thibault Cholez

Encadrant universitaire : Jean-Marie Moureaux

Remerciements

Je tiens à remercier les membres de l'équipe Madynes, tout particulièrement M. Thibault Cholez qui m'a encadré tout au long de ces 5 mois de stage. Et plus généralement, tout le personnel du LORIA pour son accueil chaleureux.

Je souhaite aussi remercier M. Jean-Marie Moureaux pour les différents conseils qu'il m'a apportés.

Table des matières

1 Introduction

1.1 Le Loria et l'équipe Madynes

Le Laboratoire lorrain de recherche en informatique et ses applications ou LORIA [9] est une unité mixte de recherche, composée du CNRS, de l'INRIA et de l'université de lorraine, travaillant sur les sciences de l'informatique. Il est le descendant du CRIN (Centre de Recherche en Informatique de Nancy), crée en 1976 suite au développement de l'informatique dans les différentes universités de Nancy. Puis dans les années 80 et 90, l'INRIA, le CNRS et le CRIN créent des équipes communes de recherches, qui ont conduit naturellement à la création du LORIA en 1997.

L'INRIA (Institut National de Recherche en Informatique et en Automatique) [6] est un établissement public qui regroupe des équipes de chercheurs travaillant dans le domaine des sciences informatiques et mathématiques. Ce sont près de 2700 collaborateurs qui travaillent dans les 8 centres, situé en France (Nancy, Rocquencourt, Rennes, Sophia Antipolis, Grenoble, Bordeaux, Lille et Saclay). Le budget de l'INRIA en 2014 était de 230 millions d'euros.

Le CNRS (Centre National de la Recherche Scientifique) [8] est lui aussi un établissement public. Avec un budget de 3,29 milliards d'euros, ce sont près de 33 000 collaborateurs regroupés en 1100 unités de recherches qui travaillent sur des sujets divers regroupés dans dix instituts.

Aujourd'hui, sous la direction de Jean-Yves Marion, le LORIA est composé de 30 équipes réparties dans 5 départements :
1 - Algorithmique, calcul, image et géométrie
2 - Méthodes formelles
3 - Réseau, systèmes et services
4 - Traitement automatique des langues et des connaissances
5 - Systèmes complexes et intelligence artificielle
L'établissement comptait en 2014, près de 176 chercheurs, 14 administratifs,27 ingénieurs et 102 doctorants représentant 48 nationalités.

L'équipe MADYNES [7] appartient au département 3, qui s'intéresse aux réseaux, systèmes distribués et parallèles et services. Les recherches de cette équipe menée par Isabelle Chrisment, portent sur la configuration, la surveillance et la sécurité des protocoles de communications et des services de l'Internet du futur. L'équipe est composée de 19 membres permanents et de plusieurs doctorants ou ingénieurs non permanents. Plusieurs domaines de recherches y sont abordés [4] :

- La sécurité : domaine incontournable dans le monde des réseaux, et de l'informatique en général, l'équipe travaille par exemple sur la détection des attaques de type *phishing*. Elle travaille également sur la gestion du trafic HTTPS, ou encore sur la surveillance des réseaux de capteurs.
- La configuration et les performances des réseaux : les performances des réseaux deviennent de plus en plus exigeantes, notamment à cause de l'importance des flux qui continuent d'augmenter. C'est pourquoi certains projets de l'équipe traitent par exemple la gestion du cache dans un réseau ICN (Information Centric Networking) ou encore l'allocation dynamique des ressources dans un réseau virtuel.
- d'autres sujets divers sont abordés, comme par exemple la conception et évaluation des architectures des réseau P2P, ou encore la modélisation et l'évaluation des performances des réseaux dynamiques.

Cette liste non exhaustive de sujets étudiés par l'équipe Madynes, montre à quel point le monde de l'Internet du futur est vaste et est en constante évolution.

1.2 Contexte

1.2.1 Le projet Doctor

Le stage effectué se déroule dans le cadre du projet DOCTOR (DeplOyment and seCurisaTion of new functiOnalities in virtualized networking enviRonments) Ce projet est un projet ANR (Agence Nationale de la Recherche) composé de plusieurs acteurs : Orange, Thales, Montimage, CNRS-LORIA, ICD (Institut Charles Delauney). Le projet a débuté en Décembre 2014 et se terminera en Décembre 2017. Ce projet dispose d'un budget de 976 271€.

L'objectif de ce projet est d'étudier la sécurité d'équipements réseaux virtuels, utilisant NFV, en prenant le cas de l'implémentation d'un nouveau protocole, NDN. Cet objectif est aussi l'occasion de pouvoir étudier comment NDN peut co-exister avec IP, pour ensuite prendre peu à peu sa place, et ainsi voir comment est-ce que l'on peut progressivement mettre en place de nouveaux protocoles.

Pour mener à bien ce but, le projet a été découpé en 5 tâches :
Tâche 0 : Gestion de projet et communication
Tâche 1 : Architecture d'un nœud virtualisé pour héberger les fonctions réseaux
Tâche 2 : Analyse de la sécurité et surveillance des architectures réseaux virtualisées
Tâche 3 : Fiabilité du réseau
Tâche 4 : Testbed et démonstration

La tâche 0 est l'étape obligatoire pour le bon déroulement du projet. En effet, c'est ici que l'on va définir les différentes réunions et les rapports réguliers qui seront effectués pour le bon suivi du projet. Elle contient aussi la phase de communication, avec le site internet [6] et les différentes conférences qui seront effectuées. Cette tâche se fait donc tout au long de la durée du projet.

La tâche 1 est dédiée à l'étude des technologies de virtualisation existantes. C'est lors de cette étape que le choix de la technologie permettant la mise en place d'un réseau virtuel, utilisant NFV, sera fait. L'objectif est de choisir un outil qui permet d'avoir un environnement permettant l'hébergement de fonctions et services réseaux, qui sera sécurisé et performant en cas de fortes charges. Par ailleurs, il faut aussi définir quels composants réseaux seront utiles pour la mise en place de l'environnement de test.

La tâche 2 est consacrée aux analyses de sécurité d'un réseau NDN virtualisé. Il faut donc déterminer les vulnérabilités et les menaces potentielles d'un réseau NDN mais aussi de la manière dont il est virtualisé. Plusieurs attaques possibles à fort impact sur le réseau seront ainsi étudiées. Cependant, pour pouvoir réaliser cette étude, il est nécessaire de pouvoir surveiller le trafic réseau de cet environnement virtuel. Pour ce faire, Montimage, va adapter son outil de monitoring, MMT, pour qu'il puisse s'intégrer à un réseau virtuel. L'étude des attaques va permettre de cibler les paramètres permettant la détection de ces dernières, et de déclencher les mesures de protections correspondantes.

La tâche 3 est la suite de la tâche 2, puisqu'il s'agit de l'étude de solutions des vulnérabilités. L'objectif est de déterminer les failles au niveau du protocole NDN, mais aussi au niveau de la manière dont il est virtualisé. Puis dans un second temps, il s'agira de mettre en place l'exécution de la contre-mesure, lorsqu'une attaque est détectée.

La tâche 4 correspond à la mise en place d'un environnement de test fonctionnel et à l'évaluation de ses performances. Cet environnement va relier l'université de Troyes et TELECOM Nancy. L'objectif est de simuler des conditions de trafic réseaux réalistes. Ainsi, il sera possible de réaliser les tests des tâches 1, 2 et 3.

Ce projet se déroule sur 36 mois, et les tâches sont réparties selon le planning suivant :

	T0	T0+3	T0+6	T0+9	T0+12	T0+15	T0+18	T0+21	T0+24	T0+27	T0+30	T0+33	T0+36
Tâche 0													
Tâche 1													
Tâche 2													
Tâche 3													
Tâche 4													

FIGURE 1.1 – Planning du projet DOCTOR

Le stage s'est déroulé entre T0+5 et T0+10

1.2.2 Enjeu du projet

Dans ce projet, deux technologies différentes sont regroupées : NFV (Network Function Virtualization) et NDN (Named-Data Networking).

NFV est une technologie montante qui est de plus en plus utilisée par les opérateurs réseaux. En effet, ces derniers cherchent attentivement les bénéfices et les possibles opportunités du déploiement de nouveaux équipements réseaux. Or ces derniers sont souvent spécifiques et donc très couteux. Le principe de NFV est de virtualiser les fonctions réseaux, qui seraient alors sous la forme d'applications qui pourraient tourner sur un serveur physique "classique". Prenons l'exemple d'un pare-feu. CISCO fourni du matériel dédié à cette tâche. Ce matériel peut s'avérer coûteux selon les performances désirées (il faut compter environ 1000€ pour un firewall 8 ports avec utilisateur illimité). NFV permet de réaliser cette fonction de pare-feu sur un serveur x86 classique tel que ceux proposés par DELL pour le même prix. De plus, le serveur x86 à l'avantage de pouvoir faire tourner d'autres applications en plus du pare-feu. Par ailleurs si l'on décide d'améliorer les capacités du pare-feu, avec le matériel de CISCO il serait nécessaire d'investir dans une nouvelle machine, tandis qu'avec le serveur, il suffit de modifier l'application virtualisée. L'objectif est donc d'utiliser le même matériel pour réaliser des actions bien différentes, ce qui permettrait aux entreprises de réaliser des économies non négligeables.

Cette technologie permet donc aux opérateurs réseaux de mettre en place de nouveaux protocoles et de nouvelles fonctions réseaux de manière progressive. Par ailleurs, l'on peut bénéficier des avantages de la virtualisation, comme par exemple l'élasticité. En effet, on peut imaginer en cas de surcharge réseau, d'instancier des éléments réseaux qui permettent de résoudre les problèmes de charges.

NDN est, quant à lui, un protocole qui a pour vocation de remplacer le protocole IP. Ce dernier n'étant plus adapté à l'utilisation d'Internet aujourd'hui, qui est plus orienté contenu. En effet, NDN se base sur l'identification des données, et non plus de leur localisation, comme avec IP. Le but est alors de décentraliser le contenu, pour qu'il soit au plus proche des demandeurs. Pour ce faire chaque nœud du réseau peut contenir des données grâce à un cache. Ainsi les données les plus demandées, seront stockées à plusieurs endroits dans le réseau les rendant plus facilement et rapidement accessibles.

1.3 Problématique

Le stage effectué, s'inscrit principalement dans la tâche 1, et plus particulièrement, dans l'étude des possibles fonctions réseaux qui composeront l'environnement de test. L'objectif est de trouver comment l'on peut tirer avantage de la virtualisation pour optimiser la fonction de routage NDN. Le protocole étant toujours en développement, les applications qui l'instancient ne sont pas encore réalisées dans le but d'être virtualisées. Notamment NFD (NDN Forwarding Daemon), qui est l'application réalisant la fonction de routage NDN.

Dans la section 2, nous étudierons au préalable le protocole NDN, l'environnement de virtualisation, ainsi que l'attaque DoS qui est une attaque très courante dans le monde IP. Ensuite, pour permettre d'augmenter les performances de la fonction de routage, nous allons voir, dans la section 3, s'il n'est pas possible de décomposer notre routeur en sous-fonctions réseaux. Sous-fonction que l'on pourrait dupliquer et ainsi utiliser l'élasticité de la virtualisation. Suite à cela, il est nécessaire de lier les différents éléments dupliqués pour pouvoir répartir la charge. Dans la section 4, nous verrons comment l'on va pouvoir mettre en place un load balancer permettant de diviser les flux entre plusieurs nœuds d'un réseau. Enfin, dans la section 5, nous verrons que grâce à ce load balancer, il sera possible de contrer une attaque DoS.

2 Etat de l'art

2.1 Le protocol Named-Data-Networking ou NDN

Notre utilisation d'Internet a beaucoup évolué depuis plusieurs années. A l'origine, Internet était un moyen de communiquer facilement entre deux sites distants, pour pouvoir échanger des informations. Aujourd'hui, Internet est plus orienté contenu (visite de site Web, visionnage de vidéos en streaming, etc...). Nous utilisons les adresses IP des différents serveurs, qui contiennent ces données, pour pouvoir les récupérer. Cependant, avec ce système de Clients/Serveurs, quelques problèmes se posent dans la recherche de contenu. En effet, la diffusion de contenu est coûteuse et les même données doivent être envoyées à chaque client individuellement, et sont ainsi répétées sur de larges portions du chemin, consommant ainsi des ressources inutilement. De plus, il est aisé de corrompre des données puisque les vérifications se font principalement sur leur localisation et les informations de connexion. On peut prendre l'exemple des mails. Même avec un serveur mail sécurisé avec une connexion chiffré TLS. Malgré toutes ces précautions, il est encore possible de recevoir des mails indésirables, tout simplement parce qu'on ne vérifie pas les données contenues dans ces mails. En effet TLS s'assure de plusieurs choses :

- L'authentification du serveur
- La confidentialité des données échangées, c'est à dire faire en sorte qu'une tierce personne puisse avoir accès au contenu de ces échanges
- L'intégrité des données, c'est à dire que TLS fait en sorte que les données ne soient pas altérées au cours de l'échange

Ces trois points ne permettent pas de s'assurer de la sécurité du contenu.

C'est à la suite de ce constat que Jacobson et al. [17] ont émis l'hypothèse de remplacer nos adresses IP par des noms, qui désigneraient non plus la localisation du contenu, mais le contenu en lui-même. Jacobson, l'inventeur du contrôle de congestion de TCP [18], a alors pris la tête du projet CCNx, qui est une application de ses recherches sur le Content-centric Networking (CCN), qui va être repris pour faire le projet NDN en 2010 [2]. Depuis que le projet a débuté, un réseau international a été mis en place regroupant plusieurs nœuds aux Etats-Unis, en Europe, en Chine et au Japon. Grâce à ce testbed, différentes applications réseaux, basées sur NDN, ont pu être testées. Le site de NDN propose par exemple des outils de messagerie instantanée, d'audioconférence ou encore de diffusion de vidéo en streaming.

2.1.1 Le principe du protocole NDN

Comme dit précédemment, ce protocole repose sur le fait que ce sont les contenus qui sont au cœur des échanges. Ainsi une donnée aura un nom spécifique qui sera sous forme de préfixe tel que :

/ndn/videos/films/StarWars

Maintenant que l'on a un moyen de spécifier une donnée, il faut propager la requête au sein du réseau pour pouvoir la récupérer. Pour ce faire, deux types de paquets existent : Les paquets de type Interest et les paquets de type Data.

Les paquets Interest sont composés de 4 champs [3] :
- Name - Contient le préfixe de la donnée, ainsi que son numéro de segment.
- Selectors - Contient plusieurs champs tels que le nom de la clé souhaitée pour signer les données (utile pour choisir un publisher particulier), ou encore un boolean permettant de demander à ce que la donnée soit la plus récente.
- Nonce - Contient une chaîne de caractère de 4 octets aléatoires. La combinaison du préfixe et de ce champ permettent d'identifier de manière unique un paquet Interest et ainsi éviter les boucles.
- Guiders - Contient la durée du paquet.

Les paquets Data sont, quant à eux, composés de 3 champs [3] :
- Name - Contient le préfixe de la donnée.
- MetaInfo - Contient divers informations sur la donnée, comme par exemple le type de contenu ou sa période durant laquelle on considère que la donnée est valide.
- Content - Contient le contenu de la donnée.

Un client ou consommateur va diffuser un paquet Interest dans le réseau pour demander une donnée qui lui reviendra sous la forme d'un paquet Data correspondant. La question est maintenant de savoir comment se déroule le routage de ses deux types de paquet.

2.1.2 Le routage dans un réseau NDN

Un routeur NDN est composé de trois éléments : le Content Store (CS), la Pending Interest Table (PIT) et la Forwarding Information Base (FIB).

Le CS est ce qui va faire la particularité du protocole NDN. Il permet de mettre en cache toutes les données qui vont passer par ce nœud, ce qui permet de délocaliser le contenu. Ainsi les données les plus populaires, seront présentes dans un très grand nombre de nœuds du réseau, et il sera donc plus aisé de les récupérer, puisqu'elles seront plus proches.

La PIT va contenir toutes les demandes qui sont en attente de réponse. Elle contient le préfixe de la donnée attendue, ainsi que toutes les interfaces qui sont intéressées.

La FIB est l'équivalent de la table de routage pour IP. C'est ce qui va permettre de diriger les paquets vers les bonnes interfaces (cf. Table 2.1).

Préfixe	Faces
/ndn/youtube/videos/0123456	25, 254, 12, 36
/ndn/image/toto	2, 45, 78, 32
/ndn/google/site/recherche/toto	13

Table 2.1 – Exemple d'une FIB

Lors du routage, deux cas sont possible, puisque le routeur peut recevoir deux types de paquets différents [18] [14] [12]. Lorsque le routeur reçoit un paquet de type Interest, son objectif est de trouver les données demandées pour satisfaire la demande.

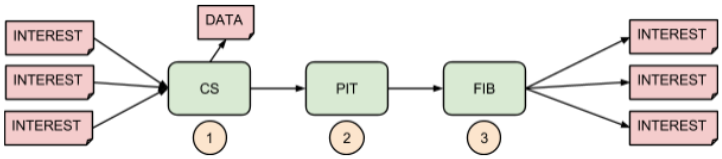

Figure 2.1 – Routage NDN lors de la réception d'un Interest

La figure 2.1 ci-dessus, montre le traitement du paquet Interest par le routeur NDN. Dans un premier temps (1), le routeur va regarder dans le CS si la donnée n'a pas déjà été mise dans le cache. Si oui, on renvoie la donnée sous forme de paquet DATA sur l'interface d'arrivée du paquet Interest reçu. Sinon, on regarde dans la PIT (2). Si la donnée a déjà été demandée, on rajoute l'interface d'arrivée de l'intérêt à l'entrée de la PIT. Sinon, on crée une entrée dans la PIT et on passe dans la FIB (3) pour trouver le chemin vers la donnée demandée. Des paquets Interest seront ensuite envoyés sur toutes les faces qui correspondent au chemin le plus précis dans la FIB.

Lorsque le routeur reçoit un paquet de type Data, il faut qu'il retransmette la donnée vers les interfaces qui l'ont demandée.

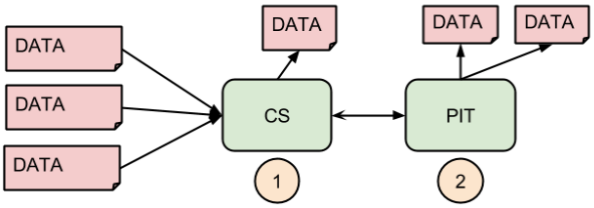

Figure 2.2 – Routage NDN lors de la réception d'une donnée

La figure 2.2 ci-dessus, montre le traitement du paquet Data par le routeur NDN. Dans un premier temps le routeur va regarder dans son CS si la donnée n'existe pas déjà (1), si c'est le cas on jette le paquet. Puis on va regarder dans la PIT s'il existe une entrée qui correspond à la donnée reçue (2). Si ce n'est pas le cas, alors le paquet est dropé, sinon, on transmet la donnée vers toutes les interfaces qui sont enregistrées dans l'entrée de la PIT correspondante et on la place dans le CS. Comme on peut le voir, la FIB n'intervient pas dans ce cas.

Prenons un exemple pour voir le déroulement du routage :

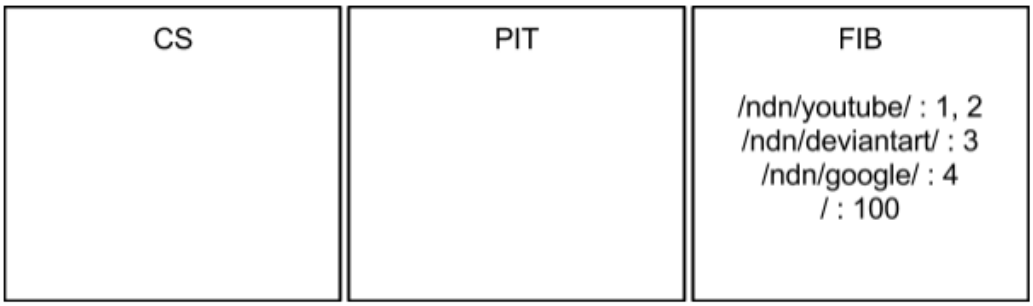

Figure 2.3 – Exemple d'un routeur NDN avant qu'il ait reçu des paquets

Supposons que le routeur reçoive un paquet Interest pour la donnée /ndn/youtube/video=Akp6514 depuis la face 15. On regarde dans un premier temps dans le cache si la donnée n'est pas présente. Ici, il n'y a rien, donc on ajoute une entrée dans la PIT et on regarde dans la FIB où on doit faire suivre le paquet. Dans la FIB l'entrée 1 correspond, donc on envoit un paquet Interest vers les faces 1 et 2. Voici donc l'état de notre routeur après réception de ce paquet :

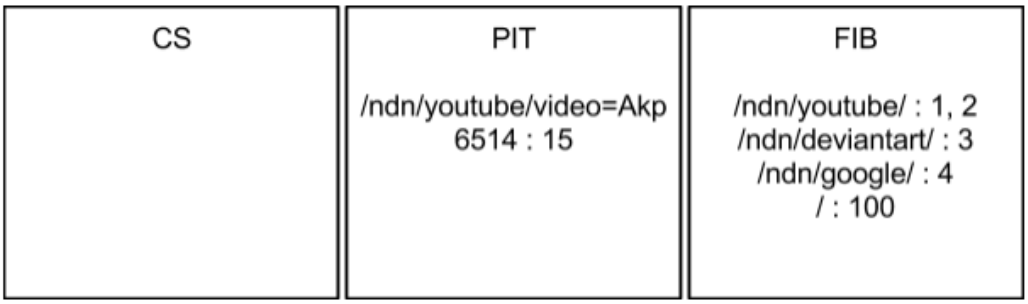

Figure 2.4 – Exemple d'un routeur NDN après réception d'un premier paquet Interest

Imaginons maintenant que l'on reçoive un paquet Interest pour la même donnée depuis la face 13. On regarde d'abord si la donnée est présente dans le cache, ce qui n'est pas le cas ici. Puis on regarde dans la PIT si une entrée correspond. Dans cet exemple, une entrée est déjà présente pour la vidéo demandée, on ajoute donc la face 13 à l'entrée existante, et l'on obtient :

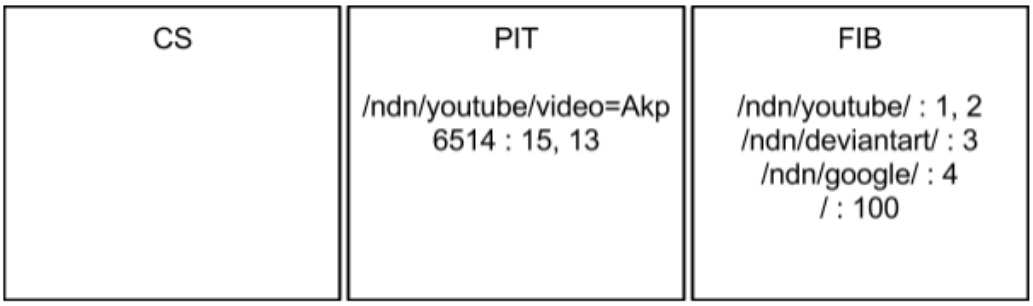

Figure 2.5 – Exemple d'un routeur NDN après réception d'un second paquet Interest

Maintenant le routeur reçoit le paquet Data correspondant à la vidéo demandée depuis la face 1. On regarde d'abord dans le content store si la donnée est déjà présente, ce n'est pas le cas ici. Puis on regarde si une entrée correspond dans la PIT. Deux face sont en attente de cette donnée, on supprime donc l'entrée, on met la donnée dans le cache et on la transfert aux faces 13 et 15. Si maintenant la même donnée arrivait depuis la seconde face, elle serait jetée puisqu'elle est déjà contenue dans le CS. Voici donc l'état du routeur après réception du paquet Data :

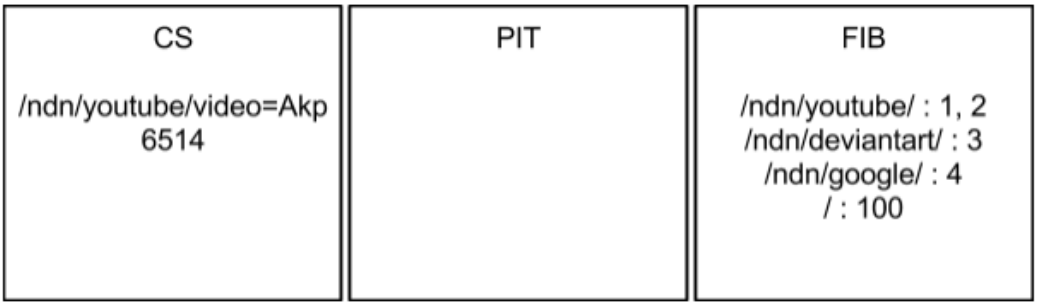

Figure 2.6 – Exemple d'un routeur NDN après réception du paquet Data

2.2 Le load-balancing

Dans le monde NDN, il n'y a encore que très peu de travaux sur la répartition de charge. Pourtant cela permettrait d'améliorer la qualité de service d'un système en divisant la charge, à l'aide d'un load balancer, sur plusieurs composants au lieu d'un seul. Notamment dans un milieu virtualisé où il est plus aisé d'instancier des éléments de réseau, et ainsi de se procurer un certains nombres de nœuds en fonction de la charge.

Un load-balancer est un élément d'un réseau qui permet de répartir la charge du trafic réseau sur plusieurs nœuds. Une application classique du load-balancer aujourd'hui est la répartition du trafic sur plusieurs serveurs. Par exemple, supposons que nous sommes administrateur d'un site Web, mais que ce dernier n'arrive plus à supporter la charge de trafic généré par les utilisateurs. Il est alors possible de dupliquer le serveur Web et d'ajouter un load-balancer pour répartir le trafic sur deux machines au lieu d'une. On obtient alors la configuration comme présenté sur la figure 2.7.

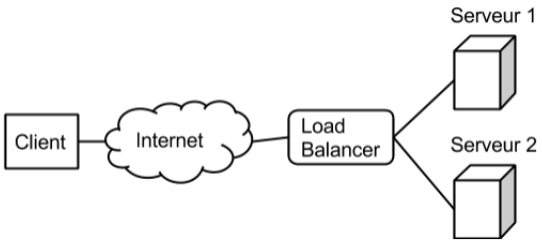

Figure 2.7 – Utilisation classique d'un load-balancer

Cette méthode est entièrement transparente pour l'utilisateur. Lorsqu'il veut atteindre le contenu souhaiter, l'adresse IP destination sera alors celle du load-balancer, qui lui, se chargera de chercher les données à renvoyer.

La question que l'on peut alors se poser est, comment le load-balancer choisit vers qui il doit rediriger les flux. Pour cela on peut distinguer deux types d'algorithmes [15] : les algorithmes statiques et les algorithmes dynamiques. Contrairement aux algorithmes dynamiques, les algorithmes statiques ne dépendent pas de l'évolution du système au cours du temps.

Voici quelques exemples d'algorithmes statiques :
- Random : Le serveur destination est choisi aléatoirement.
- Round-Robin : La charge est distribuée successivement sur chacun des serveurs.
- Weighted Round-Robin : Chaque lien vers un serveur est associé à un poids. Le nombre de connections établies vers ce serveur sera alors proportionnel au poids associé.

Voici maintenant quelques exemples d'algorithmes dynamiques :
- Dynamic Round-Robin : Cet algorithme est similaire au Weighted Round-Robin, mais les poids sont calculés dynamiquement en fonction des performances des serveurs récupérés via un outil de monitoring.
- Fastest : Le serveur choisi est celui qui à le temps de réponse le plus faible.
- Least Connection : Le serveur choisi est celui qui a le moins de connections établies.

N'ayant que très peu de documentation sur le load balancing dans NDN, il va falloir trouver un moyen de répartir la charge d'un trafic NDN sur plusieurs nœuds.

2.3 Les différentes attaques dans un réseau NDN

L'un des aspects du projet est la sécurité du réseau NDN. Il est donc important de connaître les différentes menaces qui sont possibles dans un tel réseau. On peut lister ces différentes attaques en les regroupant dans 3 groupes [13] :
- Les attaques portant atteinte à la vie privée ou *Privacy Attack*
- Les attaques visant le cache
- Les attaques de *spoofing*, où l'attaquant se fait passer pour quelqu'un d'autre
- Les attaques de type DoS (Denial of Service), cherchant à perturber le fonctionnement du système

Dans la suite nous allons étudier quelques unes des attaques les plus importantes de chacun de ces groupes.

2.3.1 Privacy Attack

Time Analysis

L'objectif de cette attaque est de déterminer le contenu d'une partie du cache d'un routeur ou encore de savoir si une donnée en particulier, est présente dans le cache d'un routeur. Pour ce faire l'attaquant va se servir du temps nécessaire pour aller chercher la donnée en question. En effet, si la donnée est présente dans le cache du routeur, il n'est pas nécessaire de passer par le réseau pour la récupérer. Le temps nécessaire est donc plus faible que si elle n'était pas présente. Prenons l'exemple du réseau suivant (2.8) :

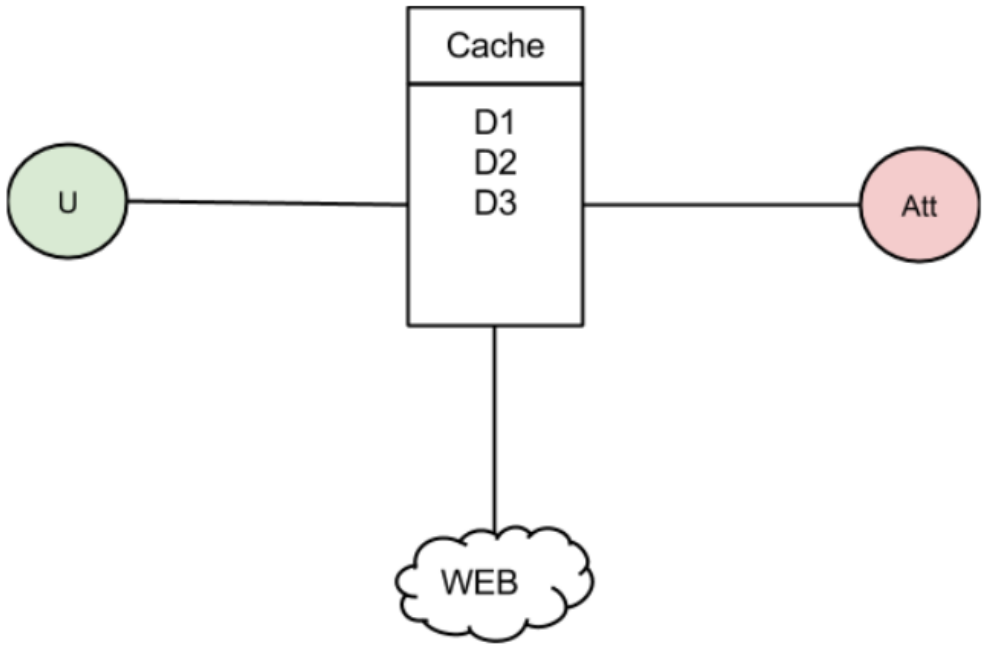

Figure 2.8 – Exemple d'un routeur ciblé par un attaquant souhaitant faire une analyse de son cache

L'objectif est de déterminer si les données D2 et D4 sont présentes dans le cache du routeur. Examinons les deux cas possibles lorsque l'on effectue une requête pour une donnée D :

- Si elle est présente dans le cache du routeur, l'attaquant envoie un paquet Interest et le routeur envoie directement la donnée sous la forme d'un paquet Data.
- Si elle n'est pas présente dans le cache du routeur, l'attaquant envoie un paquet Interest, puis le routeur va faire suivre cette requête dans le Web. Une fois que la donnée revient, le routeur l'enregistre dans le cache et retourne la donnée vers son demandeur.

Pour pouvoir différencier ces deux cas il est nécessaire d'avoir un temps de référence qui va être discriminant. Pour cela regardons le parcours de la donnée :

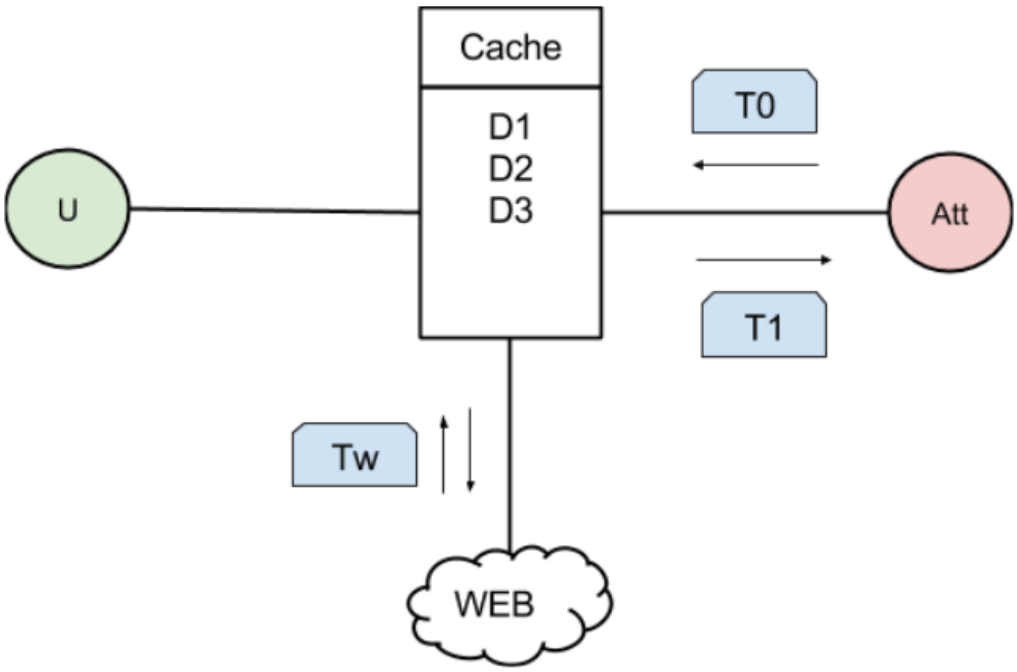

Figure 2.9 – Temps de parcours d'une requête

Le temps que met une donnée qui est présente dans le cache est donc de T0+T1 et celui d'une donnée qui n'est pas présente dans le cache est T0+T1+Tw. Le temps Tw est variable et ne permet donc pas de servir de référence. Le temps T0+T1 est fixe, et va donc nous permettre de déterminer si une donnée est présente dans le cache du routeur ciblé.

Pour déterminer le temps T0+T1, l'attaquant peut faire une moyenne de temps sur une succession de requêtes pour des données aléatoires. En effet, l'attaquant choisit une donnée aléatoire Dx, puis fait une première requête pour cette donnée. Après cela, la donnée est normalement contenue dans le cache du routeur. Il peut donc réitérer la requête qui n'ira pas dans le Web et donc qui mettra T0+T1 pour revenir. En répétant cela plusieurs fois, et en faisant la moyenne des différentes mesures obtenues il est possible de déterminer T0+T1.

En reprenant notre exemple si maintenant, on veut déterminer si D2 est dans le cache. On envoie une requête pour D2 et on regarde le temps obtenu. On obtient $T \simeq T0+T1$, on peut donc dire que D2 est présente dans le cache. On réitère l'opération pour la donnée D4, et on obtient $T > T0+T1$. On peut donc en déduire que D4 n'était pas présente dans le cache du routeur.

Watchlist et *Sniffing*

L'objectif de l'attaquant réalisant cette attaque est de filtrer une portion du réseau. Pour ce faire, l'attaquant doit avoir accès aux flux pour pouvoir surveiller le trafic. Dans le cas d'une attaque Watchlist, l'attaquant possède une liste de préfixes qu'il veut, soit surveiller soit filtrer. L'attaquant va donc supprimer les requêtes correspondant à sa liste à la volée, et peut également enregistrer les informations qu'elles contiennent. Il va aussi supprimer les paquets Data correspondants.

La figure ci-dessous représente la situation dans le réseau lorsqu'un attaquant réalise une attaque watchlist :

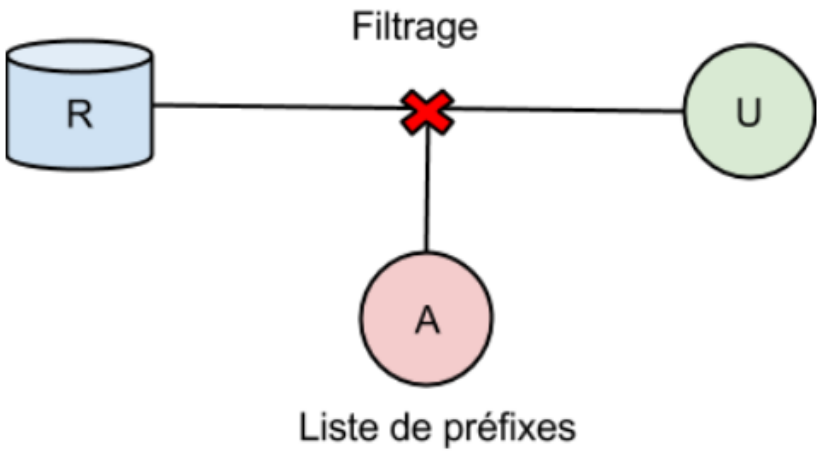

FIGURE 2.10 – Exemple d'un attaquant effectuant une attaque *watchlist*

L'attaquant s'étant inséré entre l'utilisateur et le point d'accès, il sera impossible à ce dernier d'accéder aux différents contenus listés par l'auteur de cette attaque.

Le Sniffing est une attaque similaire à une attaque Watchlist, sauf que l'attaquant ne dispose pas d'une liste de préfixes prédéfinie, mais une liste de mots clés qui vont permettre de créer dynamiquement une liste de préfixes à filtrer. Prenons un exemple :

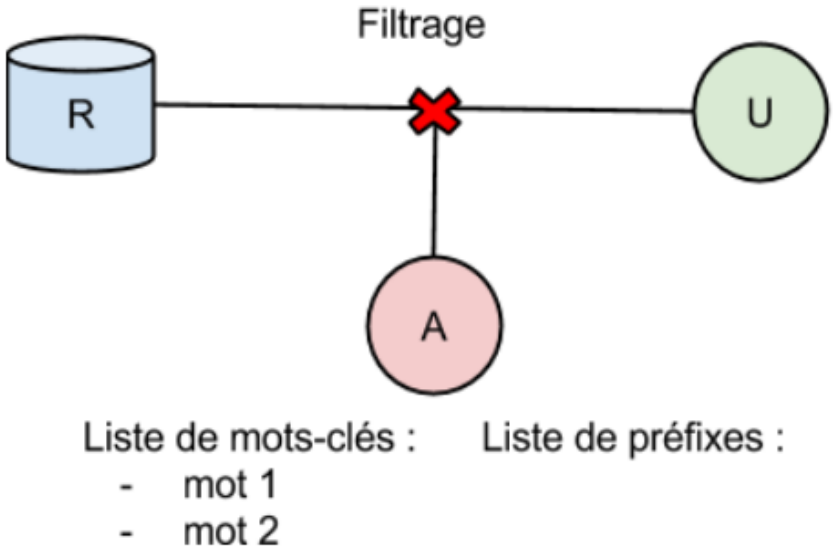

FIGURE 2.11 – Exemple d'un attaquant effectuant une attaque *sniffing*

L'attaquant possède ici une liste de deux mots clés, mot 1 et mot 2. Supposons que notre utilisateur envoie une requête pour la donnée /ndn/donneeAvecMot1. L'attaquant regarde si le préfixe correspond à un préfixe contenu dans sa liste. Ici ce n'est pas le cas, on laisse donc passer le paquet. Quand la donnée revient, on regarde son contenu, et l'on remarque que cette dernière contient un des mots clés que l'attaquant a prédéfini. On ajoute donc le préfixe à la liste de préfixes à filtrer, et on supprime le paquet. Voici donc la situation après cela :

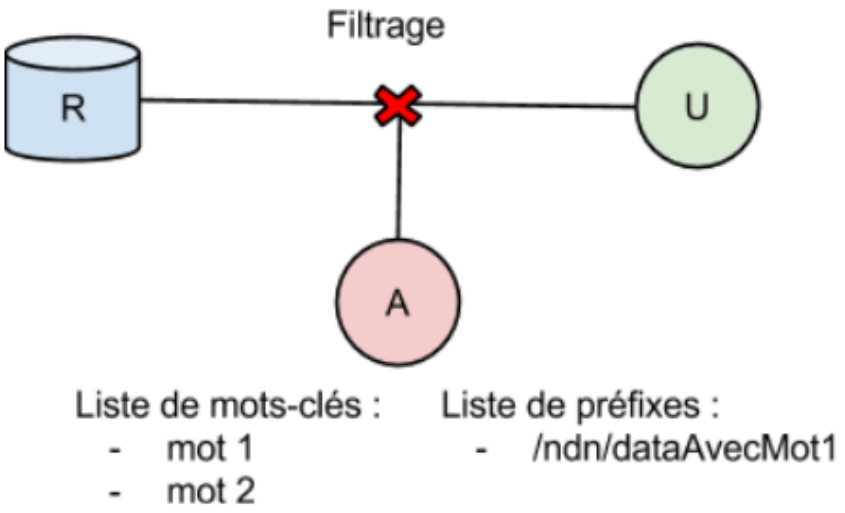

FIGURE 2.12 – Exemple de l'attaque *sniffing* après réception d'un paquet Data

La prochaine fois que l'utilisateur enverra une requête pour cette donnée, elle sera directement filtrée.

Unauthorised access

Dans cette attaque, l'attaquant souhaite accéder à du contenu destiné à un utilisateur ou un groupe d'utilisateurs en particulier. Dans le cas d'un réseau NDN, cette attaque est plus simple que dans un réseau IP, puisque les données ciblées ne sont pas localisées dans un seul endroit, mais peuvent être présentes dans plusieurs nœuds du réseau

2.3.2 Attaques liées au cache

Random request* et *Unpopular request

L'objectif de ces attaques est de modifier le contenu du cache. Prenons le cas de la première attaque, *random request*. Lors de cette attaque, l'attaquant va essayer de remplacer le contenu du cache par du contenu aléatoire. Pour ce faire, il va envoyer une succession de requêtes Interest. Une fois que les données correspondantes reviennent, elles seront mises dans le cache du routeur. Ainsi les données aléatoire vont prendre la place des données présentes, dont les données les plus populaires. Voici un exemple qui présente les conséquences d'une tel attaque :

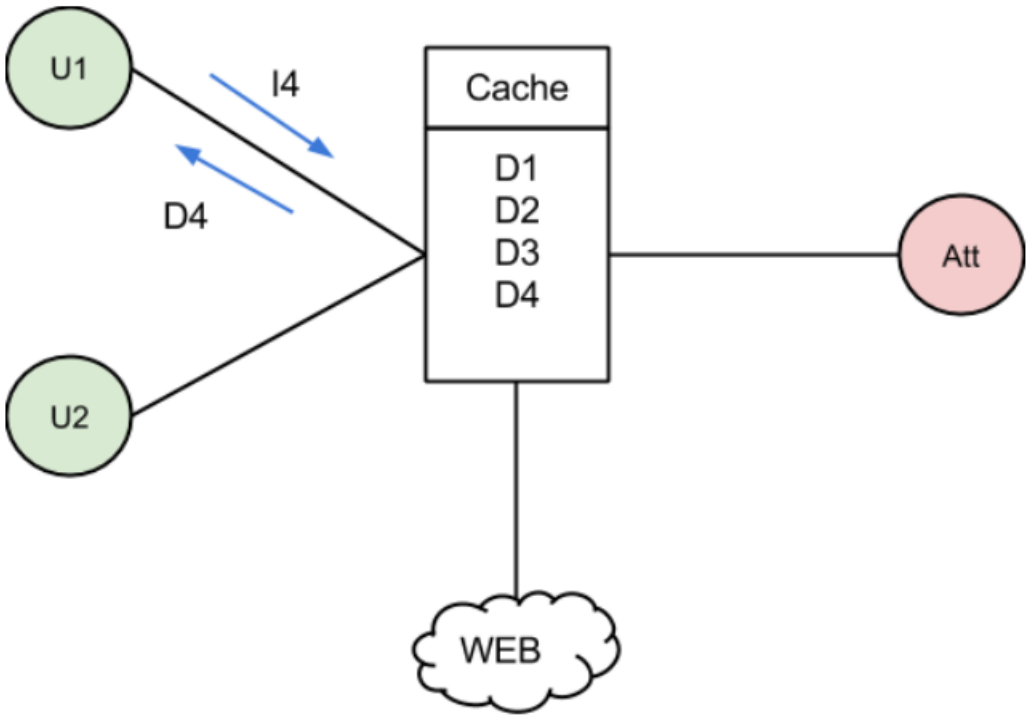

FIGURE 2.13 – Exemple d'un réseau avant l'attaque *Random Request*

Sur ce schéma, on peut voir que l'utilisateur U1 envoie une requête I4 pour la donnée D4. La donnée étant dans le cache du routeur, cette dernière revient directement vers l'utilisateur.

Regardons maintenant ce qu'il se passe lorsque l'attaquant lance ces requêtes aléatoires :

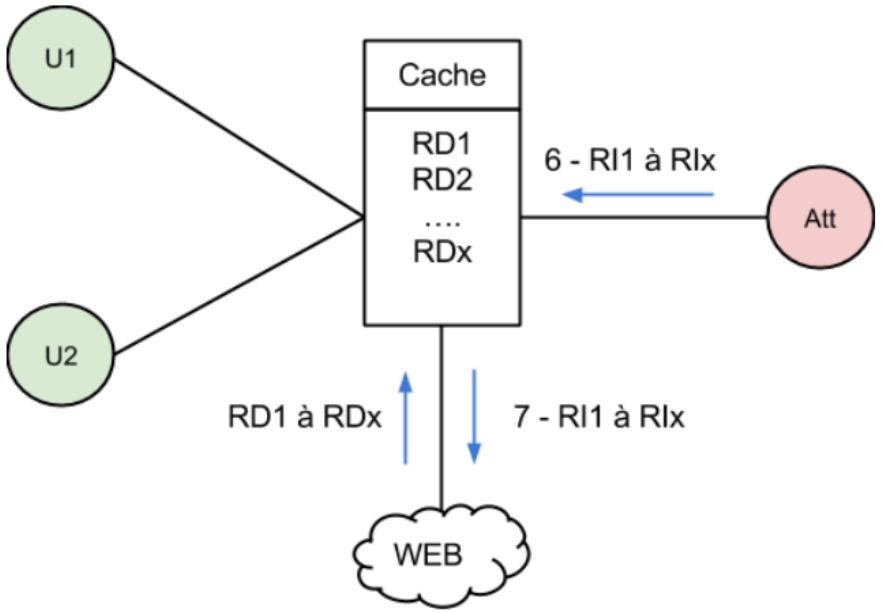

FIGURE 2.14 – Lancement de l'attaque *Random Request* par l'attaquant

Dans l'exemple, l'attaquant envoie une succession de requêtes aléatoires RI1 à RIx dans le but de les mettre en cache pour remplacer toutes les autres données déjà présentes. A l'issue de cette attaque le cache contient donc uniquement les données aléatoire demandées par l'attaquant. Si maintenant l'utilisateur U2 demande la même donnée que U1 précédemment, c'est-à-dire D4, voici ce qu'il se passe :

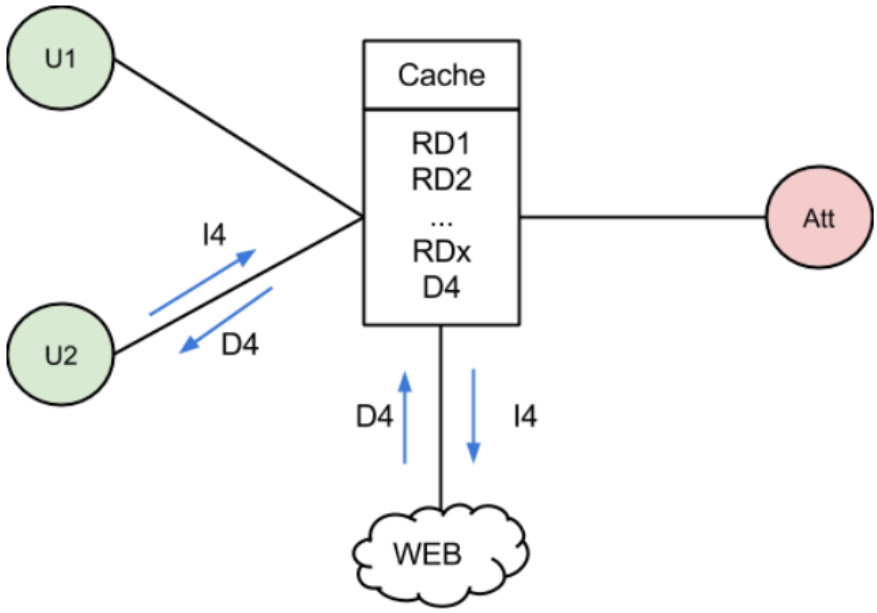

FIGURE 2.15 – Conséquence de l'attaque *Random Request*

Suite à l'attaque, la donnée D4 n'est plus présente dans le cache, il est donc nécessaire d'aller la chercher dans le réseau. Si cette attaque est donc employée régulièrement sur le routeur, l'attaquant empêche alors au routeur de stocker les données populaires.

L'attaque *Unpopular Request* est très similaire. La différence est que l'attaquant ne fait pas de requête sur des données aléatoires mais il va viser des données peu populaires. Ainsi, il va s'assurer que les données mise en cache sont peu demandées.

L'attaque *Bogus announcement*

L'objectif de cette attaque est d'empêcher la récupération d'une donnée. L'attaquant va annoncer qu'il y a une mise à jour de cette donnée avec une fréquence supérieure au temps de parcours d'une requête pour cette dernière. Ainsi le routeur va constamment chercher à avoir la donnée la plus récente, or il n'a pas le temps de récupérer la donnée mise à jour qu'il reçoit une annonce comme quoi il existe une donnée encore plus récente :

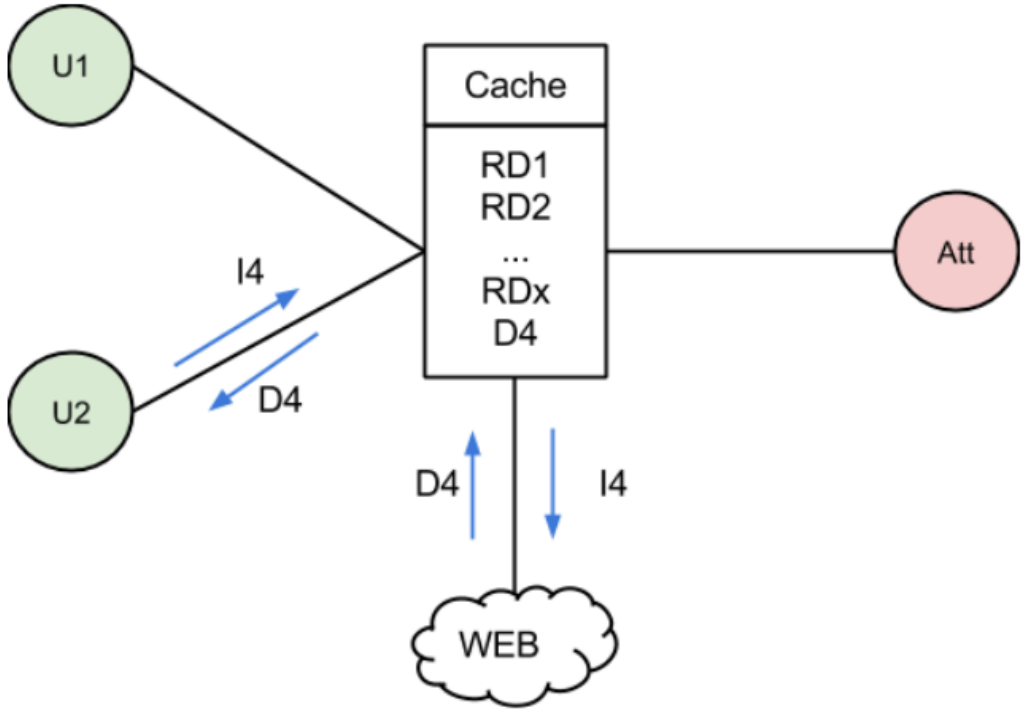

FIGURE 2.16 – Exemple d'une attaque *Bogus announcement*

Dans cet exemple, l'utilisateur envoie une requête pour la donnée D. Mais l'attaquant annonce une mise à jour de cette donnée, de tel manière à ce que le routeur n'est pas le temps de recevoir la donnée précédente. Il va donc envoyer à chaque mise à jour une requête pour la donnée D et l'utilisateur ne recevra donc jamais la donnée qu'il avait demandée.

2.3.3 Attaque de type *spoofing*

Jamming Attack

Dans cette attaque, l'attaquant se fait passer pour un subscriber honnête et va envoyer un grand nombre de paquets Interest à un routeur situé au centre d'un réseau pour maximiser le renvoi de paquets :

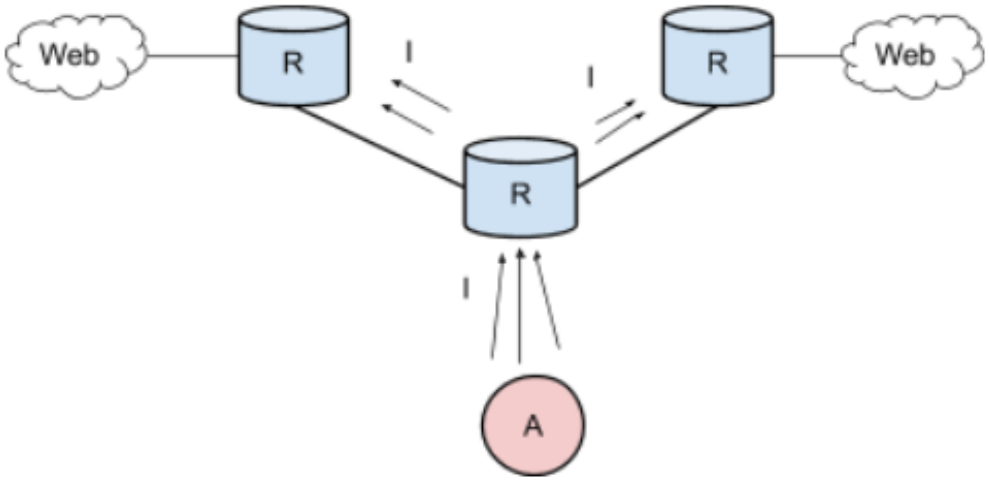

FIGURE 2.17 – Exemple d'une attaque *Jamming*

L'attaquant envoie donc un grand nombre de paquets Interest qui seront transmis de proche en proche dans le réseau. Les données qui vont revenir ne trouveront pas de destinataire, puisque l'attaquant n'est pas un véritable abonné. Le réseau est alors inondé de paquets qui seront rejetés après la fin de leur durée de vie.

Hijacking Attack

Lors de cette attaque, l'attaquant va se faire passer pour un producteur de contenu. Il va alors annoncer de fausses routes aux routeurs adjacents. Ce qui va faire que l'attaquant va récupérer les requêtes pour la donnée dont il a annoncé une fausse route, et ne va jamais répondre à ces dernières :

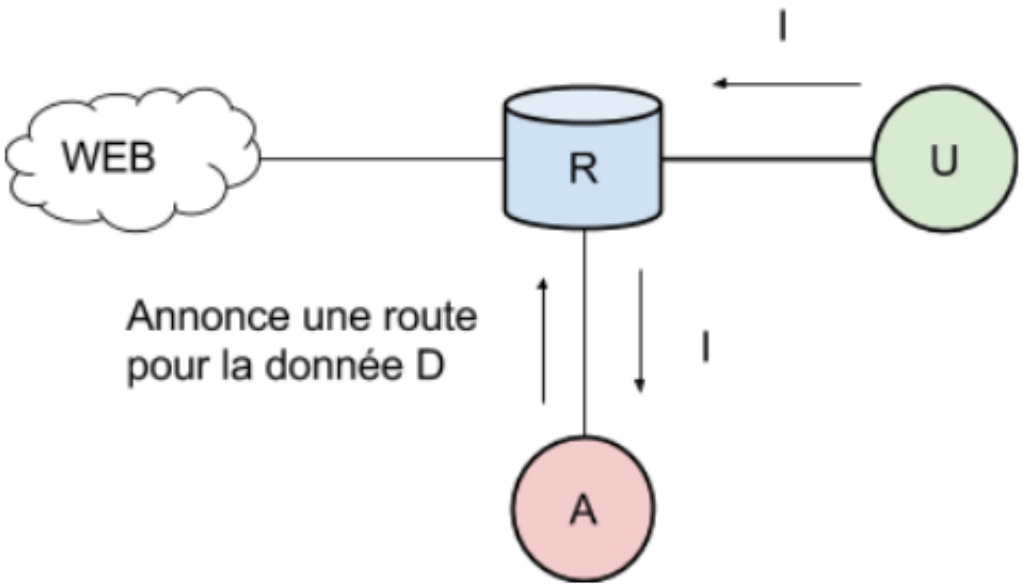

FIGURE 2.18 – Exemple d'une attaque *Hijacking*

L'utilisateur ne recevant pas sa donnée, va renvoyer une requête, ce qui va avoir pour effet d'inonder le réseau si plusieurs utilisateurs sont dans le même cas. Par ailleurs, cette attaque peut prendre une grande envergure si le réseau dispose d'un mécanisme de propagation des routes. La fausse route serait alors propagée dans le réseau et l'attaque toucherait alors un plus grand nombre d'utilisateurs.

Interception

Ici, l'attaquant va intercepter les requêtes sur un lien du réseau, en s'insérant entre deux nœuds du réseau. Prenons un exemple avec ce réseau :

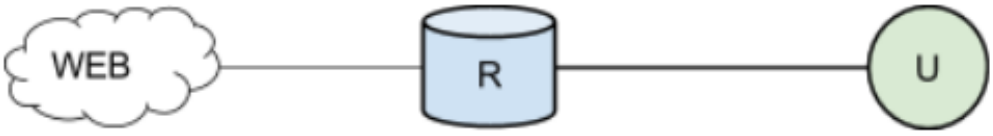

FIGURE 2.19 – Exemple d'un réseau avant une attaque de type interception

Ici l'objectif de l'attaquant va être de filtrer les paquet de l'utilisateur tout en maintenant la route. Pour ce faire, l'attaquant va se placer entre l'utilisateur et le routeur de manière transparente. Ainsi pour l'utilisateur, l'attaquant est le routeur, et pour le routeur, l'attaquant est l'utilisateur. Cette attaque est l'équivalente du *Man in the Middle* en IP.

Voici ce qu'on obtient pendant l'attaque :

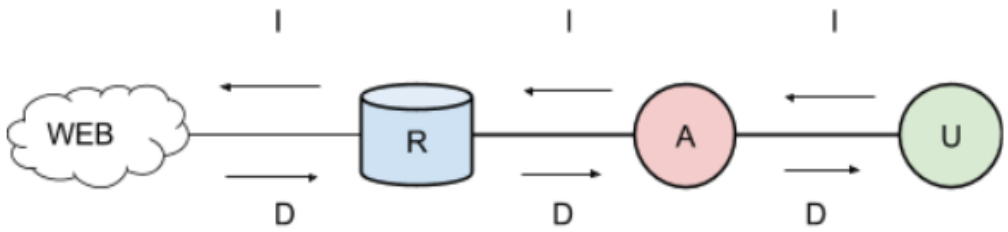

FIGURE 2.20 – Réseau subissant une attaque interception

Dans cette position, l'attaquant voit transiter tous les paquets qui passent sur le lien. Il peut alors espionner l'utilisateur ou encore filtrer ses paquets, et pourquoi pas les modifier.

2.3.4 DoS dans un réseau NDN

L'attaque la plus fréquente dans un réseau IP, est le Denial of Service (DoS). L'objectif de l'attaquant est d'empêcher le bon fonctionnement de services ou de fonctions réseaux. Cette attaque est donc tout à fait faisable dans un réseau NDN. Étant basée sur les flux importants de paquets, on peut imaginer qu'il est possible, à l'aide d'un load balancer, de rediriger le flux malveillant vers un puits, qui absorberait l'attaque.

Le moyen le plus fréquent pour réaliser cette attaque est de consommer un maximum de bande passante, pour perturber voir stopper le trafic. Pour ce faire, l'attaquant utilise plusieurs machines qu'il contrôle pour inonder le réseau. Une autre méthode utilisée par les attaquants, est d'utiliser une machine intermédiaire qui va renvoyer les paquets sur la cible de l'attaque. L'attaquant forge alors les paquets eux-mêmes pour pouvoir réaliser ce rebond.

Dans un réseau NDN, on peut distinguer deux types d'attaques DoS [16] [10] : *L'Interest Flooding* et le *Content/Cache Poisoining*.

L'objectif d'un attaquant qui fait du *cache poisoining* est de pousser un routeur à faire suivre du contenu faux ou corrompu. On considère qu'un contenu est corrompu si la signature de ce dernier est invalide. Et un contenu est faux lorsque sa signature est valide mais que cette dernière a été générée par une mauvaise clé privée. L'attaquant va donc faire en sorte de mettre dans le cache du contenu malveillant.

Alors qu'une attaque de type *cache poisoining* vise le CS du routeur, une attaque de type *interest flooding* vise principalement la PIT. Il y a trois manières de réaliser une telle attaque, en fonction du contenu que l'on va demander :

1. En faisant des requêtes sur du contenu qui existe
2. En faisant des requêtes sur du contenu qui est généré dynamiquement
3. En faisant des requêtes sur du contenu inexistant

La première méthode est peu efficace, puisque le contenu demandé va se délocaliser vers les routeurs les plus proches.

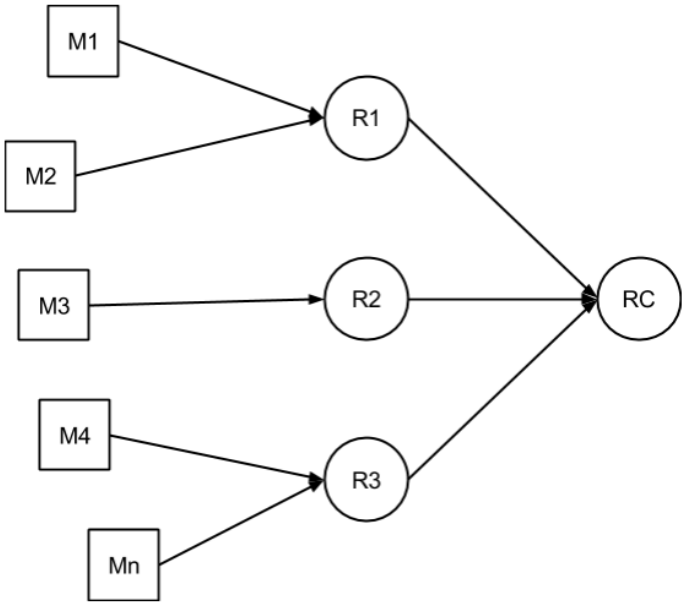

FIGURE 2.21 – Exemple d'une attaque DoS posant des problèmes pour l'attaquant

La figure 2.21 montre un exemple de configuration où l'attaquant peut rencontrer des problèmes. Supposons que l'attaquant demande à ses machines d'envoyer au routeur cible RC des requêtes vers un certain contenu. Le routeur cible va subir une forte charge, mais le contenu va être mis en cache dans les routeurs R1, R2 et R3. De ce fait le routeur cible ne recevra plus de requêtes.

La deuxième méthode quant à elle va s'attaquer non seulement au réseau mais aussi au producteur de contenu. En effet le contenu étant généré dynamiquement, à chaque requête envoyée, il faudra créer le contenu correspondant. De ce fait, le contenu ne peut pas être récupéré dans le cache d'un nœud du réseau.

La troisième et dernière méthode consiste à générer des paquets Interest de manière aléatoire, et ainsi cibler du contenu qui n'existe pas. L'objectif est alors de saturer le réseau. En effet, la requête va être transmise mais aucune réponse ne sera possible. Pour générer une requête aléatoire, l'attaquant peut très simplement demander la donnée qui correspond au préfixe /content/-randomNumber. Le préfixe étant généré de manière aléatoire, il y a une chance infime de tomber sur une donnée qui existe.

Dans tout les cas, l'objectif est de submerger le réseau de paquet Interest. Ce qui va avoir pour conséquence de remplir les PIT des routeurs concernés :

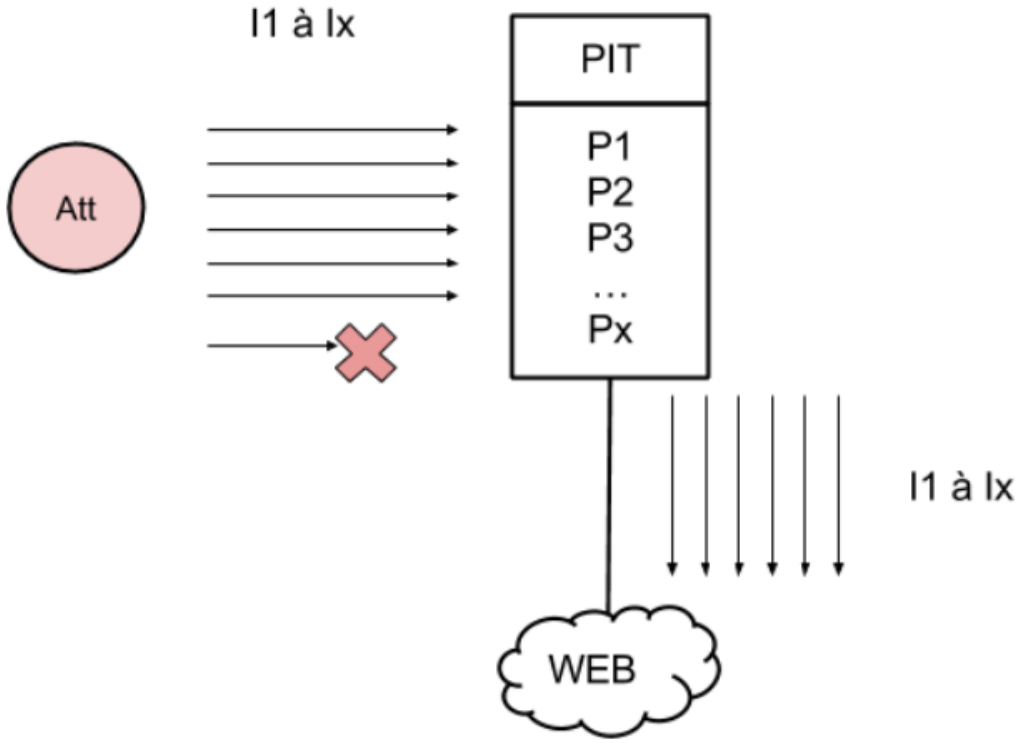

FIGURE 2.22 – Conséquence d'une attaque Dos sur la PIT d'un routeur

Comme on peut le voir sur le schéma, la PIT du routeur se remplis au fur et à mesure que les paquets Interest arrivent, jusqu'à ce que la PIT soit complètement remplis. Dans ce cas, le routeur, ne pouvant plus retenir les demandes en attente, rejette tout simplement les paquets Interest qui arrivent. Le routeur est donc hors-service, jusqu'à ce que la PIT se soit suffisamment vidée pour accueillir d'autres entrées.

Pour pouvoir éviter ce genre de désagrément, Compagno et al. [11] ont déterminé plusieurs paramètres qui permettraient de détecter une attaque de type *interest flooding* :

- Le taux d'utilisation de la PIT. En effet si le routeur reçoit rapidement un très grand nombre de paquets Interest, la PIT va progressivement se remplir jusqu'à saturation. Ce paramètre est égal à : $tauxUtilisation = \frac{QuantitdePaquetsInterestdanslaPIT}{CapacitTotaledelaPIT}$
- Le nombre de paquets Interest non satisfaits d'une interface i. Si le nombre de paquets Interest envoyés par un attaquant est élevé, la majorité d'entre eux ne sera pas satisfaite. Ce paramètre peut ce calculer de cette façon :
  $InterestInsatisfait = \#PaquetsInterestdelaFacei - \#PaquetsDataCorrespondantsReu$
- La quantité de bande passante utilisée par l'interface i pour transmettre le contenu. Si la quantité de contenu est trop importante par rapport au débit maximal du lien, il sera impossible de satisfaire toutes les requêtes à cause du timeout. Pour calculer ce paramètre on peut utiliser le ratio suivant : $Bi = \frac{NombreInterestReu}{NombreDataEnvoy}$. Les deux éléments de la fraction étant calculés pendant un intervalle donné.

2.4 Synthèse

A l'heure actuelle, le projet NDN n'a fait part d'aucun outil étant capable de répartir la charge sur les nœuds d'un réseau NDN, pour en améliorer leur qualité de service. Et il n'y a donc rien pour permettre la distribution du trafic entre plusieurs nœuds du réseau. La mise en place d'un tel outil, permettrai d'utiliser tous les avantages de la virtualisation, en ajoutant dynamiquement des nœuds dans le réseau, dans le cas d'une montée en charge. L'objectif est donc de voir si l'on peut adapter un routeur NDN pour qu'il soit adapté à un environnement virtualisé. On peut alors imaginer de découper un routeur NDN en sous-parties, et voir s'il est possible de les multiplier pour en améliorer leur performance. Par ailleurs un moyen de répartir la charge, serait de mettre en place un load balancer pour NDN. Des solutions existantes dans le monde IP pourraient alors être utilisées avec des flux NDN. Outre les aspects de performance, le système devra être robuste à l'attaque la plus répandue dans le monde IP : l'attaque DoS.

3 Conception d'une stratégie de virtualisation performante pour NDN

3.1 Décomposition d'un routeur NDN

L'objectif est de voir s'il est possible de prendre les avantages de la virtualisation, pour améliorer la qualité de service d'un routeur NDN en cas de montée en charge. Nous allons donc voir s'il est possible de décomposer notre routeur en sous-fonctions, pour pouvoir les dupliquer séparément. Pour ce faire nous allons partir du schéma global suivant, qui regroupe les deux précédents (Figures 2.1 et 2.2) :

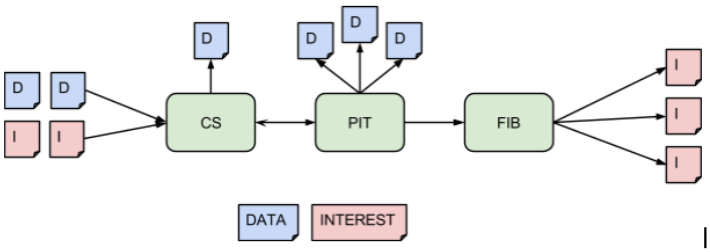

Figure 3.1 – Schéma global d'un routeur NDN

Un découpage du routeur, qui parait presque évident, est de le couper en trois parties qui sont le CS, la PIT et la FIB. L'objectif est de voir s'il n'est pas possible d'utiliser l'élasticité de la virtualisation pour pouvoir augmenter la performance du routeur. Nous allons donc faire une succession d'hypothèses pour voir quels en seraient les avantages et les inconvénients.

3.1.1 Ajout d'un CS supplémentaire

Imaginons que l'on puisse avoir 2 CS au lieu d'un seul. On obtient alors le schéma visible dans la figure 3.2 :

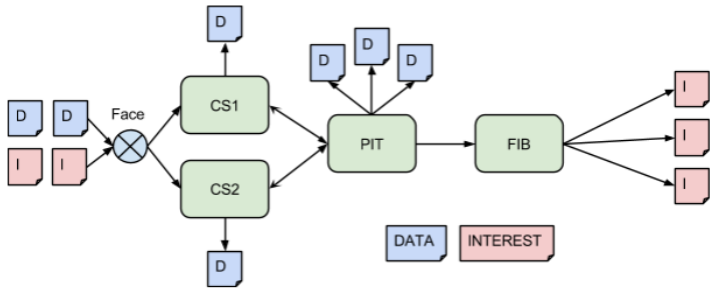

FIGURE 3.2 – Schéma de base d'un routeur NDN avec 2 CS

Le premier avantage de cette solution est qu'il est possible de répartir la charge sur 2 CS au lieu d'un seul. Par ailleurs, en cas d'atteinte de la limite de stockage d'un des CS il est possible d'utiliser le second.

Cependant cette manière de faire pose plusieurs problèmes. Tout d'abord, il est possible que des données soient dupliquées dans les CS ou que la donnée demandée soit jetée.

Par exemple, imaginons le scénario suivant :

1. Un paquet Interest arrive vers CS1 et veut la donnée D.
2. La donnée D arrive vers CS1 et est transmise puis stockée dans CS1.
3. Un autre paquet Interest arrive vers CS2 et veut la même donnée D. *(On perd du temps puisqu'elle est déjà stockée dans CS1)*
4. La donnée D revient, mais passe par CS1. D étant déjà dans CS1 le paquet est jeté.
5. Un troisième Interest arrive vers CS2 et veut récupérer D.
6. La donnée D revient et passe par CS2. CS2 ne contenant pas la donnée, elle est transmise puis mise en cache dans CS2. La même donnée est donc dans CS1 et dans CS2.

Pour pallier à ce problème, il faudrait faire en sorte de diviser les flux en plusieurs groupes qui emprunteraient toujours le même chemin dans le système.

3.1.2 Ajout d'un CS supplémentaire avec un load balancer

Voici un schéma qui présente cette hypothèse :

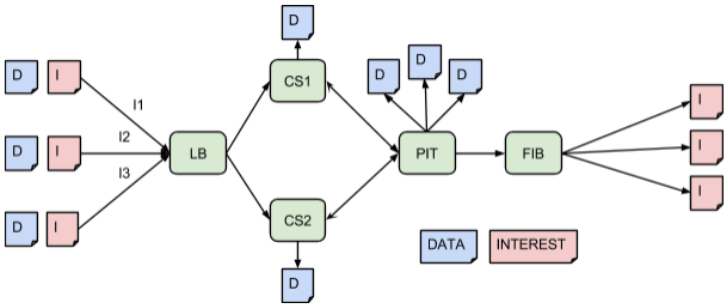

FIGURE 3.3 – Schéma d'un routeur NDN avec 2 CS et des load balancer

Pour éviter les problèmes que l'on a évoqués dans l'hypothèse 1, on met en place un système de load balancing qui va permettre de diriger un paquet de manière unique entre les deux CS. Grâce à cela on conserve les avantages évoqués pour l'hypothèse 1, et on résoudrait les problèmes de duplication de données et de perte.

On peut par exemple penser à faire ce choix en fonction de l'interface d'arrivée des paquets, mais ce n'est pas optimal puisque cela ne permettrait pas de synchroniser les données et les demandes :

0. On suppose que les interfaces I1 et I3 sont dirigées vers CS1 et I2 vers CS2.
1. Un paquet Interest arrive sur l'interface I1 et est donc dirigé vers CS1. Il veut la donnée D.
2. La donnée D arrive par l'interface I2, elle est donc transmise vers I1 puis stockée sur CS2.
3. Un paquet Interest arrive sur l'interface I3 et est donc dirigé vers CS1. Il veut la donnée D mais celle-ci ne s'y trouve pas. Il faut donc redemander la donnée pour pouvoir la retransmettre.

Dans cette situation, on est beaucoup moins efficace que dans l'hypothèse 1, puisque tous les paquets Interest demandant D qui arrivent sur une interface dirigée vers CS1, entraineront une nouvelle demande de D plutôt que de réutiliser celle du cache.

En revanche il est possible de faire le load balancing à partir du préfixe de la donnée demandée ou reçue. Dans ce cas, tous les paquets qui sont liés à une même donnée seront dirigés vers le même CS. Cela peut toutefois poser quelques problèmes de performances dans le cas d'une instanciation dynamique des composants.

Imaginons un cas simple. Nous allons utiliser un load balancer qui va diviser en deux les préfixes : les préfixes commençant par [a-m] et ceux par [n-z].

0. Nous disposons d'un routeur NDN simple, avec CS et PIT vide (CS1 - PIT - FIB)
1. Nous recevons un Interest pour /A/images/monImage.png, puis nous recevons la Data correspondant qui est donc stockée dans le CS1.
2. De même que précédemment pour la data /Z/videos/maVideo.flv.
3. CS1 contient donc 2 data /A/images/monImage.png et /Z/videos/maVideo.flv. On décide d'instancier un CS supplémentaire que l'on nommera CS2. Nous nous plaçons donc dans le cas de l'hypothèse 2. Un load balancer est donc aussi instancié. Les paquets commençant par [a-m] seront envoyé vers CS1 et les autres vers CS2.

4. Un paquet Interest arrive au niveau du load balancer pour la donnée /Z/videos/maVideo.flv. Ce paquet est donc envoyé vers CS2 qui est vide.
5. La donnée est donc à nouveau demandée et le paquet Data correspondant arrive au niveau du load balancer. Ce paquet est envoyé au niveau de CS2 et la donnée va y être stockée.
6. Au final CS1 contient : /A/images/monImage.png et /Z/videos/maVideo.flv et CS2 contient /Z/videos/maVideo.flv. Une donnée est donc contenue dans les deux CS

Pour résoudre ce problème mis en évidence par cette situation, il faudrait, au moment de l'instanciation, faire un "tri" des données qui sont contenues dans les éléments dupliqués pour les mettre "à la bonne place". Cette manipulation pourrait être très coûteuse à l'instanciation, mais permettrait peut-être d'augmenter la performance du routeur. Ce problème est réduit avec le temps dans le cas où les données ont un niveau de fraîcheur (temps au bout duquel on considère que la donnée n'est plus d'actualité).

3.1.3 Ajout d'une CS et de la PIT associée

Comme pour les deux parties précédentes, voici un schéma qui illustre cette hypothèse :

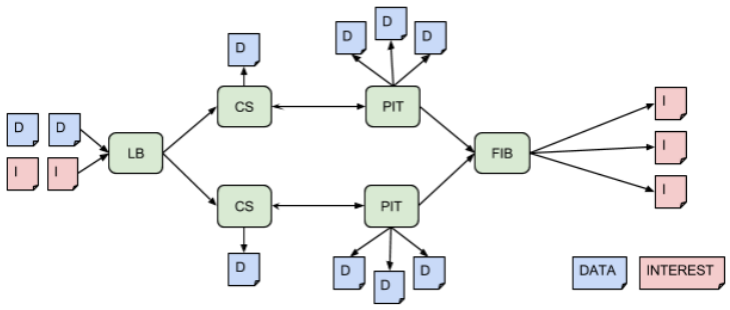

Figure 3.4 – Schéma d'un routeur NDN avec 2 CS et deux PIT

Cette configuration n'apporte que très peu d'avantage supplémentaire par rapport au cas précédent. Ayant deux PIT au lieu d'une seule, la charge à ce niveau est donc réduite.

3.2 Optimisation pour un routeur monolithique

De la même manière que dans la partie précédente, le but est d'optimiser un routeur NDN grâce aux avantages de la virtualisation. Nous allons désormais garder notre routeur en un seul morceau et voir comment il est possible de le dupliquer de manière optimale.

3.2.1 Duplication du routeur avec load balancing

Voici le schéma simple de cette configuration :

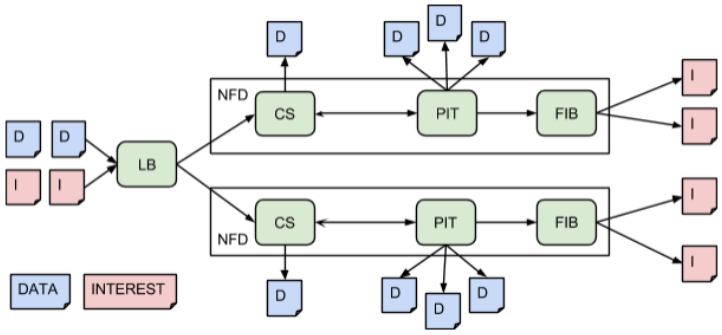

FIGURE 3.5 – Schéma d'un routeur NDN avec tous ses composants dupliqués

L'intérêt de cette configuration serait de diminuer la charge sur la FIB. Cela nous éviterait par ailleurs de toucher au démon NDN existant.

3.2.2 Duplication du routeur sans load balancing

Dans cette sous-partie nous allons voir s'il n'est pas possible de se passer du load balancer. Voici la configuration de base que nous allons étudier :

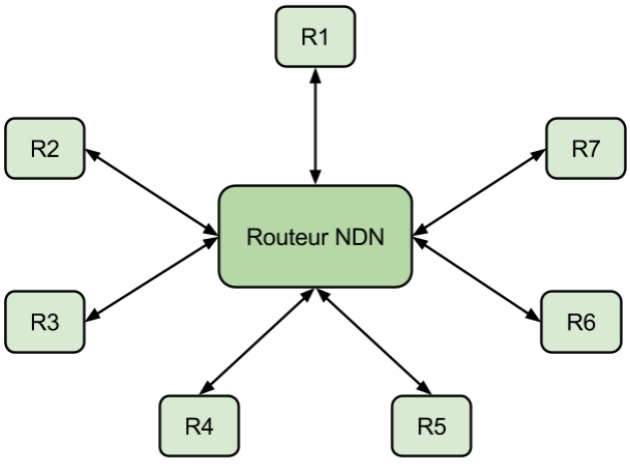

FIGURE 3.6 – Schéma de d'étude d'un routeur NDN

Nous avons donc un routeur NDN connecté à 7 autres routeurs. Par ailleurs, nous allons utiliser du routage dynamique pour faciliter la configuration des routeurs et pour permettre au réseau d'avoir une meilleure flexibilité. L'objectif va être de dupliquer notre routeur central afin de pouvoir assurer le service en cas de montée en charge.

Dans un premier temps nous allons dupliquer le routeur et créer un lien entre les deux. Ce lien permet d'accéder rapidement au cache de l'autre routeur :

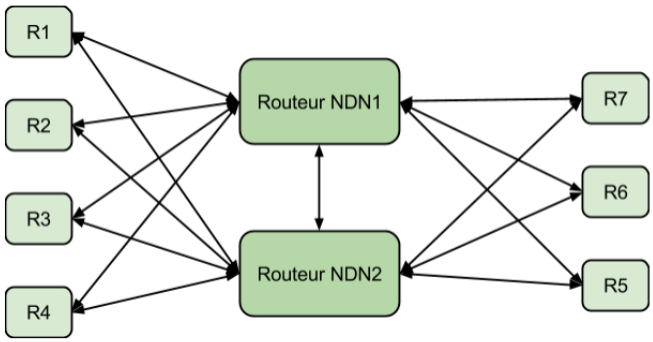

FIGURE 3.7 – Schéma d'un routeur NDN et de son duplicata relié à tous les autres routeurs

NDN fonctionne sur le principe du broadcast. On va envoyer les paquets Interest vers tous les potentiels chemins qui conduisent à une source, ce qui peut poser problème :

0. On suppose que R1 envoie un Intérêt I pour une source publié par R6.
1. Avec le routage dynamique, la FIB de R1 s'est mise à jour. L'entrée pour I contient donc au moins NDN1 et NDN2.
2. R1 envoie donc l'intérêt I vers NDN1 et NDN2.
3. NDN1 et NDN2 ont tous deux R6 comme plus proche voisin, ils envoient donc I vers R7.
4. R6 reçoit donc deux fois I. Il envoie donc la donnée correspondante D vers NDN1 et NDN2.
5. NDN1 et NDN2 reçoivent D, ils vont tous deux la mettre en cache et la renvoyer vers R1.
6. R1 va recevoir plusieurs fois la donnée D, il va prendre la première arrivée et il va dropper les autres.

Dans cette situation les deux routeurs sont mis à contribution de manière équivalente. Et plus grave, ils encombrent le réseau plus qu'ils ne l'allègent. Par ailleurs, les caches des deux routeurs vont contenir les mêmes données. Cette solution n'est donc pas envisageable.

Une autre possibilité serait d'assigner un poids différents sur les liens, pour qu'un de ces derniers ait la priorité sur les autres :

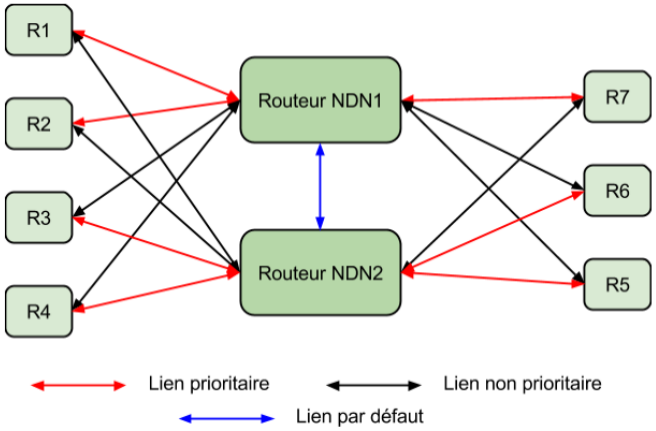

FIGURE 3.8 – Schéma d'un routeur NDN et de son duplicata relié à tous les autres avec une priorité des liens

Dans cette situation, les paquets Interest seront envoyés uniquement par le lien qui est prioritaire. Si un de nos routeurs NDN1 ou NDN2 reçoit un paquet Interest, mais que le routeur destination est relié à lui via un lien non prioritaire, le routeur fera suivre le paquet par la route par défaut. Par exemple, si R1 veut communiquer avec R3. R1 va donc envoyer un paquet à NDN1. NDN1 regarde s'il possède un lien prioritaire avec R3, ce qui n'est pas le cas, il va donc envoyer le paquet par la route par défaut, c'est-à-dire vers NDN2. NDN2 va, de son côté, regarder si il possède un lien prioritaire vers R3, ce qui est le cas, donc il peut remettre le paquet à son destinataire. Le fait que les liens prioritaires soient répartis sur les deux routeurs, permet de diviser la charge sur les routeurs. En effet, NDN1 ne prendra plus en charge les communications entre les routeurs R3, R4, R5, R6, et de meme pour NDN2 qui ne prendra plus en charge les communications entre R1, R2, R7. Cependant les données provenant de communication entre R1, R2, R7 et R3, R4, R5, R6 seront mises en cache dans les deux routeurs NDN1 et NDN2. Par ailleurs, cette configuration permet aussi d'assurer les communications en cas de perte d'un lien, puisque le second pourra prendre le relais.

3.3 Conclusion de cette étude

Suite à cette étude, on peut en conclure que la chaîne de routage n'est pas conçue pour être divisée en sous-fonctions réseaux. Le principal inconvénient étant qu'il faut retravailler le démon NDN existant pour n'obtenir que peu d'avantages. Il n'est donc pas intéressant de découper le routeur NDN. Pour éviter cela, la solution se porte donc vers la conservation d'un routeur monolithique. Deux solutions sont alors possibles : avec un load balancer ou sans. Avec load-balancer on se focalise sur les paquets, alors que sans on se focalise sur les faces. La deuxième solution nécessite un travail important lors de l'instanciation d'une nouvelle machine (création des liens et répartition des priorités). Par ailleurs, il faut modifier le démon NFD, pour qu'il prenne en compte les priorités des liens lors du routage des paquets. La première solution a donc été retenue. Il faudra donc mettre en place un load balancer qui permettra de répartir la charge sur différents nœuds du réseau.

4 Réalisation d'un load-balancer

4.1 Implémentation d'un load balancer à l'aide d'Iptables

Maintenant que le choix de la solution est établi, il faut la mettre en œuvre. Pour ce faire il faut réussir à mettre en place un load balancer.

Le choix s'est porté au départ sur un load blancer permettant de répartir un trafic réseau NDN, qui co-existerait avec IP. En effet, le fait de se baser sur IP, plutôt que sur NDN tout seul, permet de gagner en performance, puisque l'analyse d'un paquet NDN est très coûteux et le routage est donc plus lent. Le préfix d'une donnée n'est pas de longueur fixe, il faut donc lire chaque paquet pour pouvoir déterminer sa route. Pour comprendre le fonctionnement de NDN avec IP, il faut tout d'abord étudier le fonctionnent de NFD.

NFD est l'outil permettant d'effectuer le routage NDN. Pour configurer un réseau NDN, il est nécessaire de définir des Faces qui correspondent aux entrées/sorties du routeur. Ces Faces peuvent être définies de plusieurs manières : soit grâce à l'adresse MAC du destinataire, soit grâce à son adresse IP. Cette coexistence avec IP, permet de créer un pont entre un monde tout NDN et un monde tout IP, permettant d'instaurer le protocole de manière progressive, et transparente pour l'utilisateur. Après avoir défini les faces, il faut remplir la FIB en précisant le préfixe et la ou les faces correspondantes. Il est possible de consulter la liste des faces ou le contenu de la FIB grâce à la commande "nfd-status" (cf Listing 4.1).

Listing 4.1 – Résultat de nfd-status

```
General NFD status:
                    nfdId=/localhost/daemons/nfd/KEY/ksk-1429168436304/ID-CERT
                  version=0.3.4
                startTime=20150901T080902.043000
            currentTime=20150904T113444.632000
                 uptime=271542 seconds
       nNameTreeEntries=8
            nFibEntries=2
            nPitEntries=2
  nMeasurementsEntries=0
             nCsEntries=65536
          nInInterests=86151
         nOutInterests=72434
              nInDatas=134952
             nOutDatas=84407
Channels:
  unix:///run/nfd.sock
  udp6://[::]:6363
  udp4://0.0.0.0:6363
  tcp6://[::]:6363
  tcp4://0.0.0.0:6363
Faces:
  faceid=1 remote=internal:// local=internal:// counters={in={0i 134954d 0B}
      out={72436i 0d 0B}} local persistent point-to-point
  faceid=254 remote=contentstore:// local=contentstore:// counters={in={0i 0d
      0B} out={0i 0d 0B}} local persistent point-to-point
  faceid=255 remote=null:// local=null:// counters={in={0i 0d 0B} out={0i 0d 0
      B}} local persistent point-to-point
  faceid=256 remote=udp4://224.0.23.170:56363 local=udp4://152.81.13.165:56363
       counters={in={0i 0d 0B} out={0i 0d 0B}} non-local persistent multi-
      access
  faceid=257 remote=ether://[01:00:5e:00:17:aa] local=dev://eth2 counters={in
      ={0i 0d 0B} out={0i 0d 0B}} non-local persistent multi-access
  faceid=258 remote=fd://24 local=unix:///run/nfd.sock counters={in={86150i 0d
       4343814B} out={0i 84407d 36930268B}} local on-demand point-to-point
  faceid=67278 remote=fd://26 local=unix:///run/nfd.sock counters={in={3i 0d
      137B} out={0i 2d 954B}} local on-demand point-to-point
FIB:
  /localhost/nfd nexthops={faceid=1 (cost=0)}
  /localhost/nfd/rib nexthops={faceid=258 (cost=0)}
RIB:
  /localhost/nfd/rib route={faceid=258 (origin=0 cost=0 ChildInherit)}
Strategy choices:
  / strategy=/localhost/nfd/strategy/best-route/%FD%03
```

Pour réaliser un load balancer NDN sur IP, il est donc possible d'utiliser des outils existants. Le choix s'est porté vers un outil qui n'est pas prévu à cet effet au départ : Iptables [1]. Iptables est un outil permettant de contrôler Netfilter, un module du noyau Linux qui permet la gestion des paquets IP, permettant entre autre de réaliser la fonction de pare-feu. Iptables agit sur Netfilter grâce à une succession de règles. Ces dernières permettent de définir la conduite à suivre selon les paquets reçus. Par exemple, il est possible de rejeter tous les paquets provenant d'une certaine adresse IP (cf Figure 4.1) :

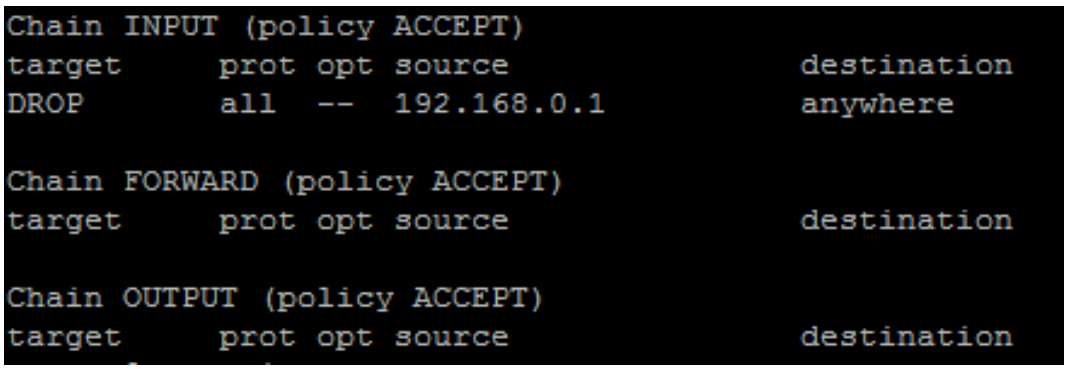

FIGURE 4.1 – Exemple d'une règle iptables rejetant tous les paquets provenant de la machine 192.168.0.1

Comme on peut le voir sur la figure 4.1, il y a trois tableaux qui correspondent à 3 chaînes : INPUT, FORWARD et OUTPUT. Un paquet va être traité en suivant la liste des règles d'une de ces trois chaînes. Si le paquet a comme destination cette machine, il suivra les règles de la chaîne INPUT. Si la source du paquet est cette machine, la chaîne appliquée sera OUTPUT. Et enfin, si la source et la destination du paquet sont toutes deux différentes de cette machines, ce sont les règles de la chaîne FORWARD qui seront appliquées. Chacune de ces chaînes est constituée d'une liste de règles. L'ordre de ces règles est très important. En effet elles s'effectuent dans l'ordre. Prenons l'exemple présenté sur la figure 4.2.

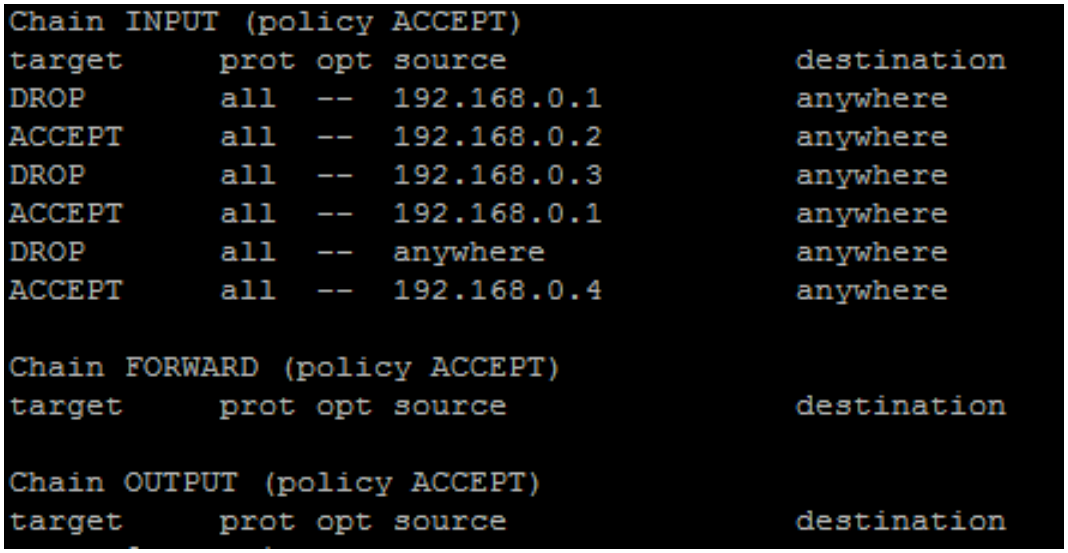

FIGURE 4.2 – Exemple d'une liste règle iptables

Décortiquons le comportement des règles appliquées sur cette machine :
Règle 1 - On rejette tous les paquet provenant de 192.168.0.1
Règle 2 - On laisse passer tous les paquet provenant de 192.168.0.2
Règle 3 - On rejette tous les paquet provenant de 192.168.0.3
Règle 4 - On laisse passer tous les paquet provenant de 192.168.0.1
Règle 5 - On rejette tous les paquet
Règle 6 - On laisse passer tous les paquet provenant de 192.168.0.4

Supposons maintenant que l'on reçoit un paquet provenant de 192.168.0.2. On regarde la règle 1. Comme elle ne correspond pas on passe à la règle 2, qui nous dit de laisser passer le paquet. Si maintenant la machine reçoit un paquet provenant de la machine 192.168.0.4, les règles 1 à 4 ne correspondent pas. C'est donc la règle 5 qui est appliquée, et le paquet est rejeté. La règle 6 est alors inutile. Il faut donc faire très attention à l'ordre des règles qui est souvent une source d'erreur.

L'avantage avec iptables c'est qu'il est possible de faire du prérouting et du postrouting. C'est-à-dire, qu'il est possible de réaliser des opérations avant ou après que la machine ai réalisé la procédure de routage. La figure 4.3 montre le déroulement des opérations à chaque arrivée de paquet :

Figure 4.3 – Succession des opérations lors de l'opération de routage

Ce qui nous intéresse dans la réalisation d'un load balancer, c'est de faire du prérouting. En effet, l'objectif est de changer l'adresse ip destination du paquet reçu pour qu'il soit redirigé vers une autre machine.

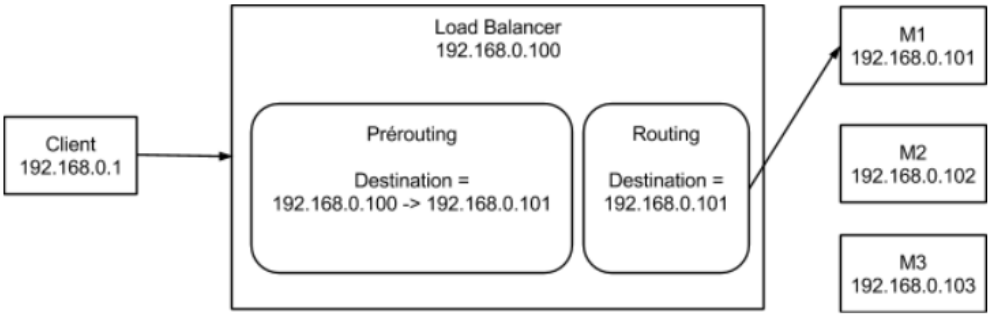

Figure 4.4 – Opération de prérouting pour réaliser un load balancer

La Figure 4.4, montre ce que l'on doit obtenir. Le paquet doit être dirigé sur l'une des machines M1, M2 ou M3. Notre client va envoyer un paquet avec pour destination le load balancer (192.168.0.100). Une fois reçu, on passe dans la phase de prérouting, et on change l'adresse ip destination du paquet par l'adresse ip de M1 (192.168.0.101). Lors de la phase de routage, le load balancer voit l'adresse de M1, et il redirige donc le paquet vers cette dernière.

A ce stade, le load balancer agit comme un simple routeur, puisqu'il redirige les paquets toujours vers la même machine. Il faut maintenant alterner entre les machines pour répartir le trafic. Pour cela, il existe une option permettant d'appliquer une règle tous les x paquets. Voici la configuration de iptables permettant de réaliser un load balancer, utilisant un algorithme de round-robin, entre les machines M1, M2 et M3 :

Listing 4.2 – Configuration du load balancer avec iptables

```
1  Chain PREROUTING (policy ACCEPT)
2  target     prot opt source               destination
3  DNAT       udp  --  anywhere             anywhere             udp dpt:6363
       state NEW statistic mode nth every 3 to:192.168.0.101:6363
4  DNAT       udp  --  anywhere             anywhere             udp dpt:6363
       state NEW statistic mode nth every 2 to:192.168.0.102:6363
5  DNAT       udp  --  anywhere             anywhere             udp dpt:6363
       state NEW statistic mode nth every 1 to:192.168.0.103:6363
6
7  Chain INPUT (policy ACCEPT)
8  target     prot opt source               destination
9
10 Chain OUTPUT (policy ACCEPT)
11 target     prot opt source               destination
12
13 Chain POSTROUTING (policy ACCEPT)
14 target     prot opt source               destination
```

C'est grâce à l'option "every x" que iptables peut se comporter comme un load balancer. En reprenant l'exemple précédent, si un client envoie un paquet sur le load balancer, il sera redirigé vers M1, puisque la première règle est vérifiée. Si un second client envoie lui aussi un paquet vers le load balancer, il sera lui redirigé vers M2. En effet la première règle n'est pas vérifiée puisque cette dernière a été appliquée la fois précédente. C'est donc la seconde règle qui est effectuée.

Par ailleurs, iptables conserve en mémoire les connexions, c'est-à-dire que des paquets provenant de la même connexion seront toujours dirigés vers le même nœud. Ainsi si l'on demande une vidéo composé de 100 segments, toutes les requêtes Interest et Data générés emprunteront le même chemin. Après un certain temps sans paquets, le load balancer oublie cette connexion. Dans le but de simplifier la configuration de iptables, un script a été écrit (cf Annexe A).

4.2 Evaluation des performances de la solution choisie

4.2.1 Environnement expérimental

Dans le but de tester les performances du load balancer, un environnement virtuel a été mis en place grâce au logiciel Docker [5]. Docker permet d'instancier des containers dans lequel il est possible d'exécuter des applications dans un environnement isolé. L'objectif est de simuler l'exécution du logiciel, comme s'il se trouvait sur une machine standard. Les différents containers se partagent alors, les performances du matériel de la machine hôte.

Par ailleurs, Xavier Marchal, lui aussi stagiaire sur le projet Doctor, a réalisé un outil permettant de faire des tests de performance du trafic NDN. Cet outil, permet de générer un trafic important de paquet NDN et en mesure le débit. C'est lui qui va permettre de réaliser l'évaluation du load balancer.

La figure 4.5 présente le schéma du test mis en place :

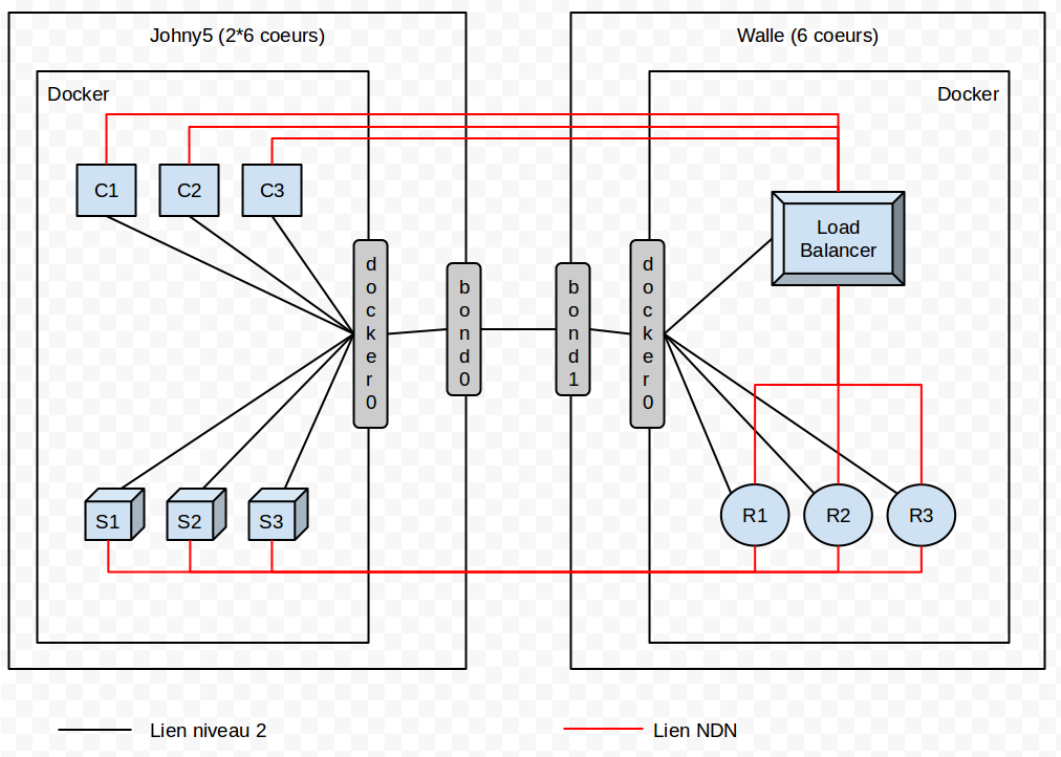

FIGURE 4.5 – Schéma de l'expérience mise en place pour le test du load balancer

Deux serveurs sont mis à contribution : Walle et Johnny5. Johnny5 va héberger les containers qui joueront les rôles de clients et serveurs et Walle va contenir les routeurs et le load balancer. Le but est d'isoler les routeurs et le load balancer pour qu'il n'y ait pas d'interférence avec les clients et les serveurs, et ainsi que le test soit plus fiable.

Il y a en tout 6 clients et 6 serveurs, dont le trafic sera réparti sur 6 routeurs au maximum (pour des questions de lisibilité, il n'y a que 3 clients, serveurs et routeurs sur le schéma 4.5). A chaque test, le nombre de routeur sera incrémenté de 1, et l'on pourra voir ainsi l'impact du load balancer sur la quantité de paquet NDN traité et sur le pourcentage de CPU utilisé.

Chaque élément sera contenu dans un container docker différent. Tous les containers sont reliés au docker0, qui est lui-même relié à l'interface de sortie bond0 ou bond1 de la machine physique correspondante.

Chaque client va envoyer des intérêts successifs pour les données ayant pour préfixe /testx/y, où x est le numéro du client et y est un nombre qui est incrémenté à chaque envoi d'un paquet Interest. Tous les paquets Interest sont alors envoyés vers le load balancer, qui va les répartir sur les différents routeurs disponibles. Les routeurs contiennent la route de tous les préfixes /testx, et redirigent alors les paquets vers le serveur correspondant. Le serveur 1 produit le contenu ayant pour préfixe /test1, etc...

L'objectif est alors de surcharger les routeurs disponibles afin de voir si l'on améliore les performances du routage dans le cas où l'on ajoute un nœud.

4.2.2 Résultats

Après avoir réalisé l'expérience décrite précédemment, voici les résultats obtenus :

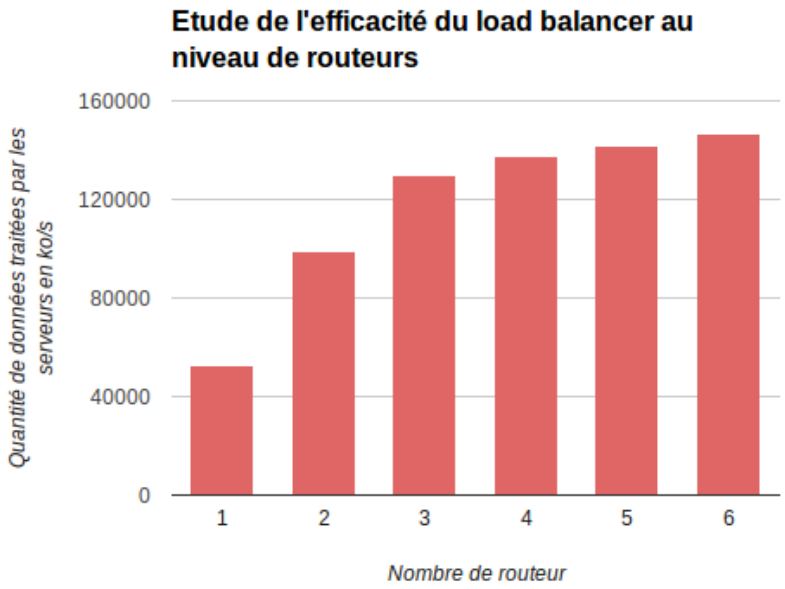

FIGURE 4.6 – Résultats : Quantité de données traités en fonction du nombre de routeurs disponibles

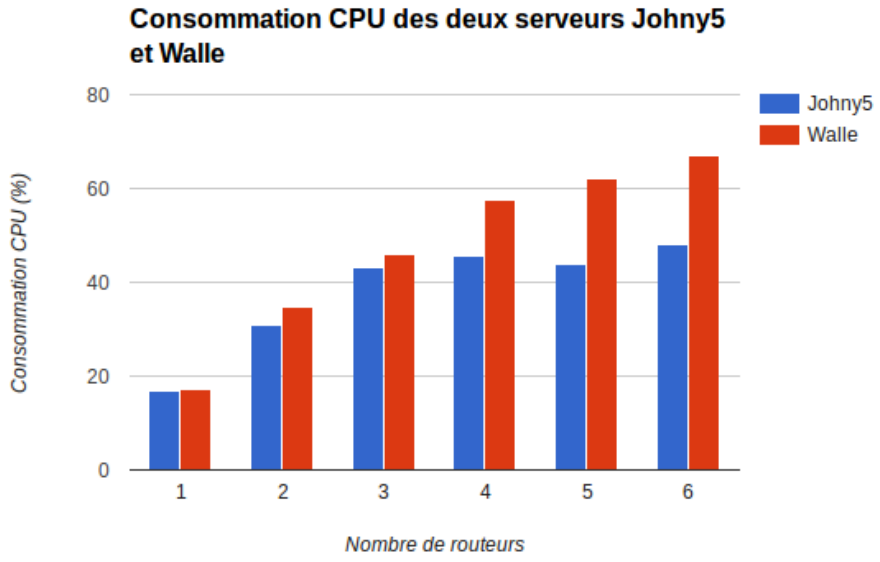

FIGURE 4.7 – Résultats : Consommations CPU des serveurs en fonction du nombre de routeurs disponibles

Comme on peut le voir sur la Figure 4.6 avec 1 seul routeur, la quantité de paquets traités représente un débit de 52 Mo/s. Après l'ajout d'un routeur, comme le trafic est divisé sur deux routeurs, la quantité de paquets traités a quasiment doublé. Puis plus on ajoute de routeur plus la quantité de données augmente, jusqu'à atteindre un plafond. Ce plafond correspond à la limite du logiciel utilisé pour le test, ainsi qu'au fait que le serveur Walle n'ayant que 6 cœurs, commence à devoir partager le temps processeur entre les différents processus qui tournent.

En ce qui concerne la consommation CPU, Johnny5 atteint un palier qui correspond au palier atteint par la quantité de données traitées. C'est en effet le seul élément sur ce serveur qui est à l'origine de cette consommation. Pour Walle, c'est le fonctionnement des routeurs qui consomme du CPU. A chaque routeur supplémentaire, il y a un container supplémentaire et donc la consommation du CPU augmente. On peut remarquer que cette augmentation est moins importante à partir du palier. En effet la quantité de paquet à router augmente peu et donc la consommation du CPU aussi.

Le load balancer remplit donc son rôle en répartissant le trafic sur plusieurs nœuds du réseau. La charge est alors divisée et les performances du système sont augmentées. Les limites sont alors les liens réseau physique qui ont un maximum de bande passante, et la quantité de CPU disponible qui limite aussi le nombre de nœuds que l'on peut instancier.

5 Mise en oeuvre de la solution dans le cadre d'une attaque DoS

Maintenant que le load balancer est opérationnel, nous allons voir s'il est possible de l'utiliser comme moyen de se défendre contre une attaque DoS.

Dans un premier temps il faut pouvoir détecter que le système subit une attaque. Plusieurs paramètres permettent de détecter qu'une possible attaque est en train de se produire (cf 2.3) :
- Le taux d'utilisation de la PIT
- Le taux de paquets Interest non satisfaits
- La quantité de bande passante utilisée pour transmettre le contenu

Seuls, ces paramètres ne sont pas la preuve qu'une attaque est en train de se produire. Il est tout à fait possible qu'un grand nombre de paquets Interest arrivent, sans que cela soit dû à une attaque. Le fait de mettre un seuil, sur cet unique paramètre, engendrerait beaucoup de faux positifs.

En revanche, il est tout à fait possible de combiner les paramètres, pour éviter de faire des erreurs. Compagno et al. ont pris l'exemple de deux paramètres pour pouvoir détecter une attaque DoS : L'espace de la PIT utilisé par les paquets Interest arrivant sur l'interface et le ratio entre le nombre de paquets Interest qui arrivent et le nombre de paquets Data qui repartent. Après avoir défini un seuil pour ces deux paramètres, il est possible de dire qu'une interface transporte les paquets d'un attaquant potentiel.

Après avoir détecté qu'une face fait l'objet d'une attaque, deux solutions peuvent être envisagées : soit on bloque le trafic provenant de la face suspectée de faire transiter l'attaque, soit on redirige le trafic vers un puits. Ces deux actions n'affecteront pas uniquement le trafic de l'attaquant, mais tout le trafic provenant d'une face du routeur. En effet, NDN n'a aucune notion de source, puisque les paquets transitent de proche en proche. Il est donc impossible d'isoler le flux malveillant. Nous allons supposer pour la suite que nous avons à notre disposition tous les outils nécessaires pour la détection d'une attaque. Au sein du projet c'est la sonde MMT et l'orchestrateur qui vont permettre de réaliser cette opération.

Pour pouvoir illustrer le concept, nous allons nous baser sur le schéma suivant :

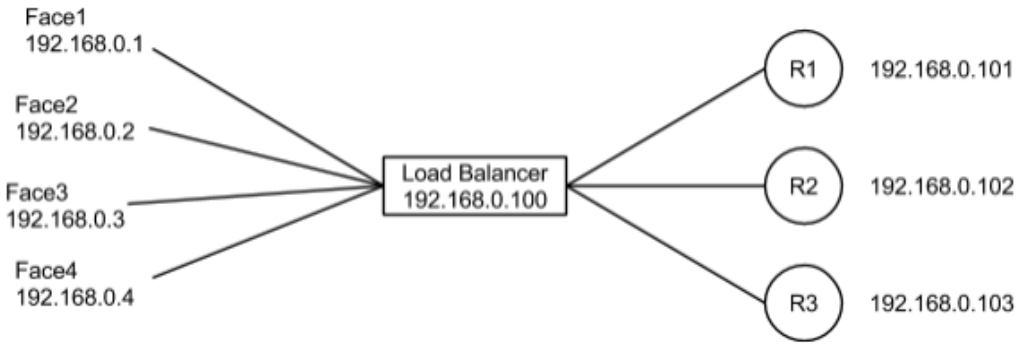

FIGURE 5.1 – Schéma de base pour l'étude de contre-mesure d'une attaque DoS

La configuration de iptables dans le load balancer est alors la suivante :

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination



Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       udp  --  anywhere             anywhere             udp dpt:6363 state NEW statistic mode nth every 3 to:192.168.0.101:6363
DNAT       udp  --  anywhere             anywhere             udp dpt:6363 state NEW statistic mode nth every 2 to:192.168.0.102:6363
DNAT       udp  --  anywhere             anywhere             udp dpt:6363 state NEW statistic mode nth every 1 to:192.168.0.103:6363

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```

FIGURE 5.2 – Configuration de base du load balancer, avec les règles de préroutages et les règles appliquées pendant le routage

L'objectif est de voir comment, à l'aide de notre load balancer, il est possible d'effectuer une contre-mesure de l'attaque DoS. Dans un premier temps, nous allons voir comment bloquer le trafic provenant d'une face suspecte. Supposons que l'on ait détecté une attaque provenant de la Face 2. L'orchestrateur va donc lancer la contre-mesure. Pour cela il peut ajouter une règle dans iptables pour bloquer le trafic provenant de l'adresse 192.168.0.2, qui correspond à la face 2. La commande permettant d'ajouter cette règle est :

iptables -A INPUT –source 192.168.0.2 -j DROP

Cette commande permet simplement de filtrer le trafic dont la source est 192.168.0.2. La figure 5.3 montre la configuration du load balancer après application de cette contre-mesure.

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  192.168.0.2          anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination



Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       udp  --  anywhere             anywhere             udp dpt:6363 state NEW statistic mode nth every 3 to:192.168.0.101:6363
DNAT       udp  --  anywhere             anywhere             udp dpt:6363 state NEW statistic mode nth every 2 to:192.168.0.102:6363
DNAT       udp  --  anywhere             anywhere             udp dpt:6363 state NEW statistic mode nth every 1 to:192.168.0.103:6363

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```

FIGURE 5.3 – Configuration du load balancer après ajout de la règle permettant de bloquer l'attaque

La règle ajoutée dans la chaine INPUT, permet de dire que tout le trafic provenant de l'adresse 192.168.0.2 est rejeté. Après cela, l'orchestrateur pourra retirer cette règle lorsque l'attaque ne sera plus effective pour permettre au trafic de reprendre normalement.

Il est aussi possible de rediriger le trafic vers un puits, qui absorberai l'attaque, et où il y aurait possibilité d'étudier l'attaque par exemple. Pour ce faire le load balancer doit ajouter une règle permettant de rediriger le trafic provenant de la face suspectée vers ce puits.

Voici le nouveau schéma du réseau :

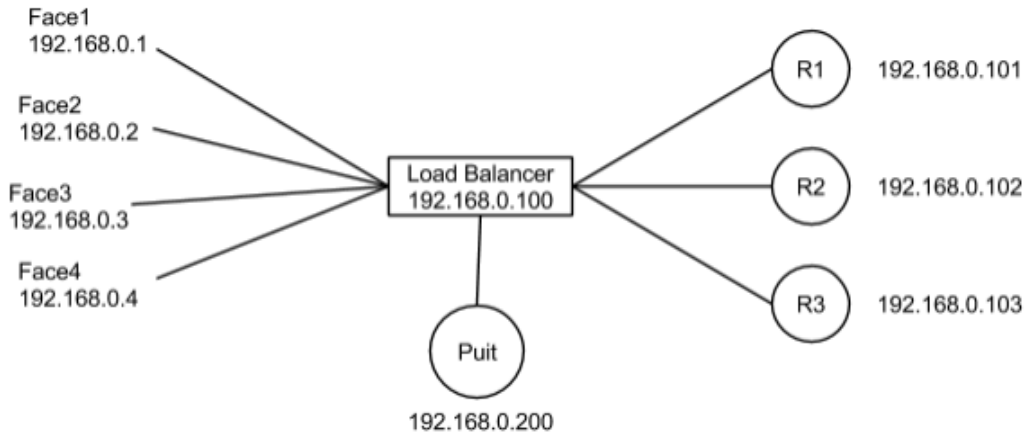

FIGURE 5.4 – Schéma du réseau avec l'ajout d'un puits pour accueillir le trafic des attaques DoS

Supposons maintenant que l'orchestrateur ait détecté une attaque provenant de la face 192.168.0.3. La règle qu'il faut donc ajouter se trouvera dans la chaine de PREROUTING, avec les règles de base du load balancer. Il faudra donc faire attention à mettre la règle de contre-mesure en première position pour qu'elle soit appliquée avant les autres. La commande permettant l'ajout d'une telle règle est :

iptables -t nat -I PREROUTING 1 -p udp –source 192.168.0.3 –dport 6363 -j DNAT –to-destination 192.168.0.200 :6363

La partie "-I PREROUTING 1" permet de dire qu'il faut ajouter la règle dans la chaine prerouting à la première position.

Voici la configuration obtenue :

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  192.168.0.2          anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination



Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       udp  --  192.168.0.3          anywhere             udp dpt:6363 to:192.168.0.200:6363
DNAT       udp  --  anywhere             anywhere             udp dpt:6363 state NEW statistic mode nth every 3 to:192.168.0.101:6363
DNAT       udp  --  anywhere             anywhere             udp dpt:6363 state NEW statistic mode nth every 2 to:192.168.0.102:6363
DNAT       udp  --  anywhere             anywhere             udp dpt:6363 state NEW statistic mode nth every 1 to:192.168.0.103:6363

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
```

FIGURE 5.5 – Configuration du load balancer après redirection du trafic provenant de 192.168.0.3 vers le puits (192.168.0.200)

La règle étant ajoutée en première position elle sera effectuée en priorité par rapport aux trois autres règles présentes.

Iptables permet donc de se prémunir contre les attaques DoS NDN, lorsqu'il coexiste avec IP. Il permet soit de bloquer le trafic de l'attaque soit de le rediriger. Cependant la méthode employée est radicale. En effet lorsque l'on bloque ou que l'on redirige le flux malveillant, on touche aussi les flux légitimes. Or il est très difficile en NDN de différencier les flux puisqu'il n'y a aucune notion de source, et on ne peut donc pas définir d'où le flux malveillant provient. Une autre méthode serait tout de même souhaitable et il faudrait donc faire une étude plus avancée sur la gestion des flux en NDN.

6 Conclusion

6.1 En résumé

L'objectif de ce stage était de trouver un moyen d'optimiser la fonction de routage NDN dans un environnement virtualisé. Le projet DOCTOR se basant sur NFV, la première idée était de décomposer un routeur NDN en plusieurs sous-fonctions réseaux. Cependant, cette méthode pose de nombreux problèmes : duplication ou rejet de paquets. Par ailleurs, il faut modifier le démon NFD pour que ce dernier soit adapté à cette utilisation. Le choix s'est donc porté vers la multiplication de routeurs relié à un load balancer.

Il a donc fallu chercher un moyen de réaliser ce load balancer. Le choix s'est donc tourné vers iptables. Cet outil, principalement utilisé pour réaliser la fonction de pare-feu, permet également de réaliser des opérations avant que le routage ne soit appliqué. Il est alors possible de simuler le comportement d'un load balancer. Après avoir réalisé un script pour faciliter la configuration du load balancer, des tests de performances ont été réalisés. Ces derniers montrent que le fait de diviser la charge sur plusieurs routeurs au lieu d'un seul, permet d'en améliorer sa performance.

Suite à cette étude, nous avons imaginé une autre utilisation de ce load balancer par l'intermédiaire de l'exemple d'une attaque DoS. Cette attaque se caractérise par l'arrivée importante de flux, qui engendre des perturbations dans le bon fonctionnement des fonctions réseaux. Le load balancer, qui traite les flux de donnée et dont la fonction principale est d'agir en tant que pare-feu, peut alors contrer cette attaque. Deux solutions sont alors possibles : on peut bloquer le flux malveillant, ou le rediriger vers un puits. Trois paramètres sont alors utiles pour détecter une attaque : le taux de remplissage de la PIT, la quantité de paquets non satisfaits et l'utilisation de la bande passante. Une fois que l'attaque est détectée par la sonde, l'orchestrateur peut alors déclencher la contre-mesure qui consiste à modifier les règles dans iptables, afin de bloquer ou de rediriger le flux ciblé.

Suite à cette étude, et grâce au load balancer, il est donc possible d'améliorer les performances d'un réseau NDN virtualisé et de se protéger contre certaines attaques telles que l'attaque DoS.

6.2 Méthodes de travail et gestion de projet

Pour mener à bien ce projet, des réunions régulières ont été réalisées, pour exposer l'avancement et les différents résultats. La figure 6.1 présentent la répartition des différentes étapes du stage pendant ces 5 mois.

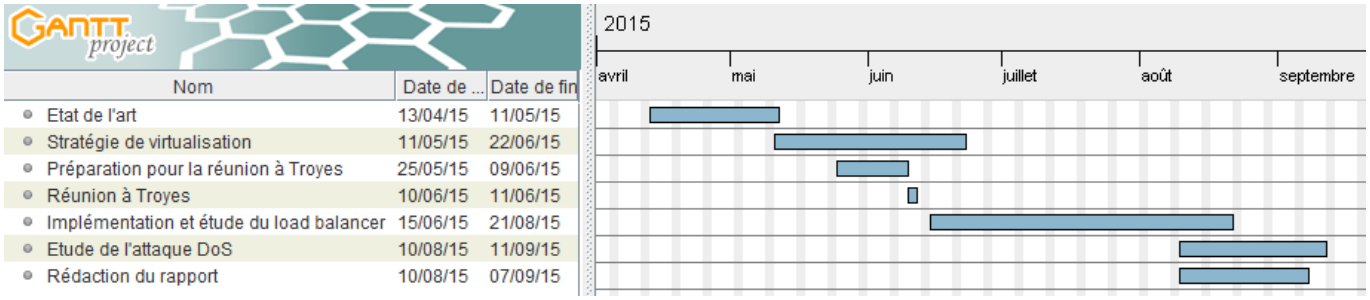

FIGURE 6.1 – Diagramme de Gantt du stage réalisé

Par ailleurs, j'ai pu participer à une des trois réunions physiques annuelles du projet DOCTOR en juin. Lors de cette réunion, les différents acteurs ont présenté leur travail, et les objectifs à court-termes ont pu être définis.

6.3 Perspectives

Pour poursuivre les résultats de ce stage, et parfaire aux objectifs du projet DOCTOR, plusieurs éléments peuvent être étudiés. Tout d'abord, il faudrait réussir à mettre en place un load balancer de niveau 2, pour qu'il puisse être utilisé avec du NDN sans IP. Ainsi, on pourrait finir la transition entre un monde tout IP vers un monde tout NDN. Par ailleurs, il faudrait mettre en place un testbed avec les sondes MMT et l'orchestrateur pour pouvoir tester les contre-mesures d'une attaque DoS. Ceci pourra être testé, dans des conditions réalistes, avec le testbed qui sera mis en place entre TELECOM Nancy et Troyes. Le trafic "normal" sera alors généré par les utilisateurs, et il sera alors possible de générer un trafic perturbateur pour voir si le réseau réagit.

Bibliographie / Webographie

[1] Documentation ubuntu sur iptables. http ://doc.ubuntu-fr.org/iptables. 36

[2] Named data networking. www.named-data.net. 7

[3] Ndn packet format specification 0.2-alpha-3 documentation. www.named-data.net/doc/ndn-tlv/index.html. 8

[4] Projets et rapport d'activité de l'équipe madynes en 2014. http ://raweb.inria.fr/rapportsactivite/RA2014/madynes/uid0.html. 2

[5] Site de docker. https ://www.docker.com/. 39

[6] Site de l'inria. www.inria.fr. 1, 2

[7] Site de l'équipe madynes. madynes.loria.fr. 2

[8] Site du cnrs. www.cnrs.fr. 1

[9] Site du loria. www.loria.fr. 1

[10] I. Moiseenko E. Uzun A. Afanasyev, P. Mahadevan and L. Zhang. Poseidon : Mitigating interest flooding ddos attacks in named data networking, 2013. 22

[11] Paolo Gasti Gene Tsudik Alberto Compagno, Mauro Conti. Interest flooding attack and countermeasures in named data networking, August 2013. 24

[12] Lan Wang Beichuan Zhang Lixia Zhang Alexander Afanasyev, Cheng Yi. Scaling ndn routing : Old tale, new design, July 2013. 9

[13] Mohammad Zulkernine Eslam G. AbdAllah, Hossam S. Hassanein. A survey of security attacks in information-centric networking, 2015. 12

[14] J. Burke V. Jacobson J. D. Thornton D. K. Smetters B. Zhang G. Tsudik kc claffy Dmitri Krioukov D. Massey C. Papadopoulos T. Abdelzaher L. Wang P. Crowley E. Yeh L. Zhang, D. Estrin. Named data networking (ndn) project, November 1988. 9

[15] Don MacVittie. Intro to load balancing for developers, June 2010. https ://devcentral.f5.com/articles/intro-to-load-balancing-for-developers-ndash-the-algorithms. 12

[16] Ersin Uzun Lixia Zhang Paolo Gasti, Gene Tsudik. Dos and ddos in named data networking, 2013. 22

[17] J. D. Thornton M. F. Plass N. H. Briggs R. L. Braynard V. Jacobson, D. K. Smetters. Networking named content, Décembre 2009. 7

[18] Michael J. Karels Van Jacobson. Congestion avoidance and control, November 1988. 7, 9

Liste des illustrations

Liste des tableaux

Glossaire

CS = Content Store : Élément d'un routeur NDN permettant la mise en cache des données qu'il reçoit

Docker : Logiciel permettant de réaliser un environnement virtuel, dans lequel on peut créer des containers hébergeant des machines virtuelles

DoS = Denial of Service ou attaque par déni de service : Attaque visant à perturber le bon fonctionnement du réseau en inondant par exemple le réseau de requêtes inutiles.

FIB = Forwarding Information Base : Elément d'un routeur NDN équivalent à la table de routage en IP. Elle permet de rediriger les paquets Interest vers le nœud suivant

Iptables : Logiciel permettant de traité les paquets IP. Il s'utilise notamment pour jouer le rôle de pare-feu.

Load balancer : Élément d'un réseau permettant de partager les flux sur plusieurs autres nœuds du réseau. Cela permet par exemple de distribuer la charge sur deux serveurs au lieu d'un seul.

NDN = Named-Data Networking : protocole imaginé par Van Jacobson, qui vise a remplacer IP. Ce protocole est centré sur les données plutôt que leur localisation.

NFD = NDN Forwarding Daemon : Logiciel permettant de simuler un routeur NDN sur une machine classique.

NFV = Network Function Virtualization : Technologie visant à virtualiser les éléments d'un réseau. Ce qui permet de rendre le réseau évolutif et plus économique.

PIT = Pending Interest Table : Element d'un routeur NDN permettant de garder en mémoire toutes les demandes en attente, ainsi que l'endroit où il faudra renvoyer la donnée.

TLS = Transport Layer Security : Protocole de sécurisation des échange Internet. Utilisé notamment pour les pages https et pour les échanges avec un serveur mail.

Annexes

A Script shell pour configurer le load balancer

Voici le script permettant de configurer iptables :

Listing A.1 – Script shell pour configurer iptables

```
1  #!/bin/bash
2
3  printUsage(){
4          echo "Usage : ./iptables_loadbalancer.sh [-d dport] [-p <udp|tcp>]
               ipDe$
5          echo "Exemple : ./iptables_loadbalancer.sh -d 12563 -p udp
               10.0.1.2,10.$
6          exit 1
7  }
8
9  nbArg=$#
10
11 protocol=udp
12 dport=6363
13 ipDest=""
14
15 #RÃ©cuperation des options
16 while getopts "d:p:" o
17 do       case "$o" in
18         d)       dport="$OPTARG"
19                  nbArg=$(($nbArg-2));;
20                  p)       protocol="$OPTARG"
21                  nbArg=$(($nbArg-2));;
22         [?])     printUsage;;
23         esac
24 done
25
26 #Il n'y a pas le bon nombre d'argument
27 if [ $nbArg -ne 1 ]; then
28         printUsage
29 fi
30
31 #RÃ©cupÃ©ration des adresses IP destination
32 shift $(($OPTIND-1))
33 ipDest=$@
34
35 #Nombre d'adresse Ip destination
36 nbD=`echo $ipDest | grep -o "," | wc -c`
37 nbD=$(($nbD/2))
38 nbD=$(($nbD+1))
39
40 #Mise en place des rÃ¨gles
41 echo -e "Flush iptables...\n"
42 iptables -t nat -F
43
```

```
44 for i in `seq 1 $nbD`;
45 do
46         ipD="`echo $ipDest | cut -d',' -f$i`"
47         echo "iptables -t nat -A PREROUTING -p $protocol --dport $dport -m
               state --state NEW -m statistic --mode nth --every $(($nb-$i+1)) --
               packet 0 -j DNAT --to-destination $ip$
48         iptables -t nat -A PREROUTING -p $protocol --dport $dport -m state --
               state NEW -m statistic --mode nth --every $(($nbD-$i+1)) --packet
               0 -j DNAT --to-destination $ipD:$dp$
49 done
50
51 echo -e "Rules done...\n"
52 iptables -t nat -L
```

Résumé

Ce stage, effectué au sein de l'équipe de recherche MADYNES du LORIA, s'inscrit dans le cadre du projet ANR, DOCTOR. Ce projet a pour but d'étudier la sécurité des équipements réseaux virtuels utilisant NFV, en prenant l'exemple de l'implémentation d'un nouveau protocole : Named-Data Networking (NDN). L'objectif du stage était de proposer un moyen d'améliorer l'élasticité et la robustesse de la fonction de routage NDN en tirant parti des avantages de la virtualisation. Le protocole NDN a pour vocation de remplacer IP. Son principe est de nommer les données grâce à un préfixe. Au fur et à mesure que la donnée parcourt le réseau, elle est gardée en cache dans les différents nœuds visités. Une solution envisagée est de découper le routeur NDN en sous-fonctions réseaux, donnant ainsi la possibilité de les dupliquer pour améliorer leur performance. Il s'avère que la chaîne de routage n'a pas été conçue pour être modifiée, et le choix s'est donc tourné vers la conservation d'un routeur monolithique. Au final, la solution du load balancer s'avère être la plus simple et efficace à réaliser. Pour réaliser le load balancer, le logiciel iptables a été étudié. Bien qu'il ait été réalisé dans le but de jouer le rôle de pare-feu, il est possible de lui donner des règles lui permettant de jouer le rôle de load balancer. Une étude de ses performances a été réalisée et il s'avère que le load balancer permet bien d'améliorer la qualité de service du routage NDN mais aussi de le rendre plus robuste. Par ailleurs, ce load balancer peut aussi jouer le rôle de barrière en cas d'attaque DoS. On peut, grâce à lui, soit bloquer le trafic malveillant, soit le rediriger vers un puits.

Mots-clés : Named-Data Networking, virtualisation, robustesse, attaque DoS

Abstract

This internship took place into the MADYNES's research team of LORIA. This work forms a part of the ANR project, DOCTOR. The aim of this project is to study the security of virtual network equipments using NFV, by taking the exemple of the implementation of a new protocol : Named-Data Networking (NDN). The goal of the internship is to suggest a way to improve elasticity and robustness of the NDN routing function, by using the advantage of virtualisation. NDN aims for replacing IP. Its principle is to name data thanks to a prefix. As soon as data packets travel across the network, data is cached into the different visited nodes of the network. A solution is to decompose the NDN router into small network functions, which allow us to duplicate them in order to improve the network performance. However, the routing chain is not implemented with the aim of being changed, and we decided to choose a solution with an entire NDN router. The solution with a load balancer is the easier solution to implement. In order to make this load balancer, the iptables software has been studied. Even if this software was created with the aim of playing the role of firewall, it can also play, with the right rules, the role of a load balancer. One performance test has been made and it reveals that the load balancer improves the NDN routing quality of service and makes the network more reliable too. Otherwise this load balancer is able to foil a possible DoS attack, Indeed, with this load balancer, it is possible to block malicious flow or to redirect it toward a black hole.

Keywords : Named-Data Networking, virtualisation, robustness, DoS attack